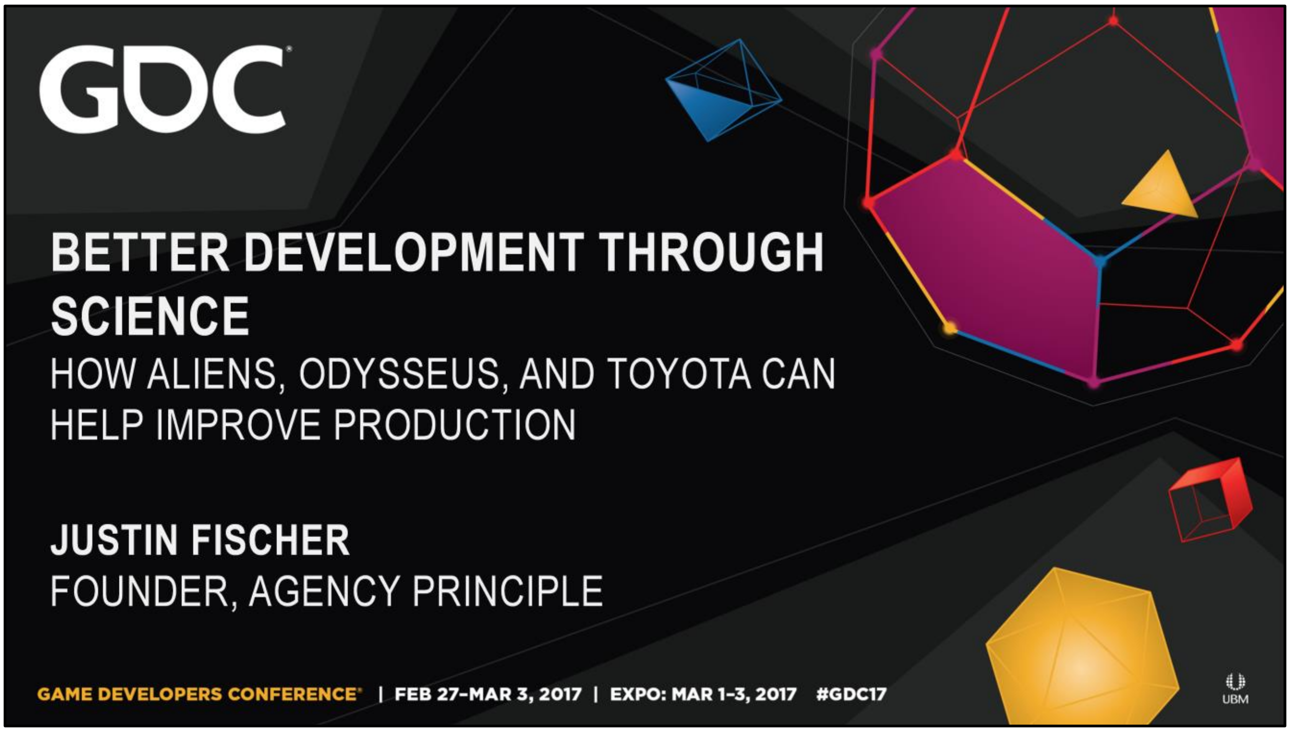


GDC

BETTER DEVELOPMENT THROUGH SCIENCE

HOW ALIENS, ODYSSEUS, AND TOYOTA CAN HELP IMPROVE PRODUCTION

JUSTIN FISCHER

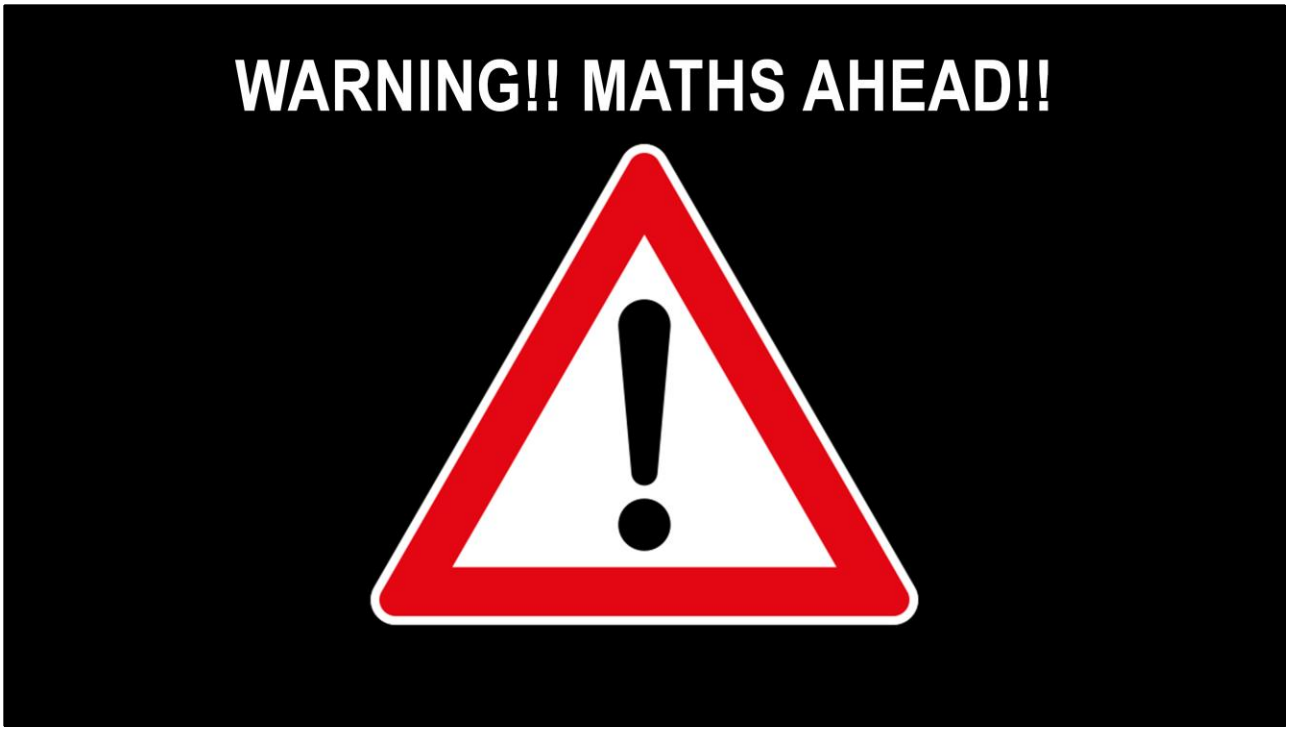
FOUNDER, AGENCY PRINCIPLE

GAME DEVELOPERS CONFERENCE® | FEB 27–MAR 3, 2017 | EXPO: MAR 1–3, 2017 #GDC17

UBM

A PREFACE...

This presentation is about how to improve production efficiency and predictability, and eliminate waste. In order to successfully communicate the high-level concepts, I need to first establish some fundamentals.



That means I need to give you a crash course in operation science. So:

- Don’t be afraid of math
- Don’t get too hung up on terminology
- Focus on the high-level takeaways and feel free to email me if you are confused about anything!

HANG IN THERE!

Hang in there! The nuts & bolts of the first half is to set the stage for the pay-off of the second half.

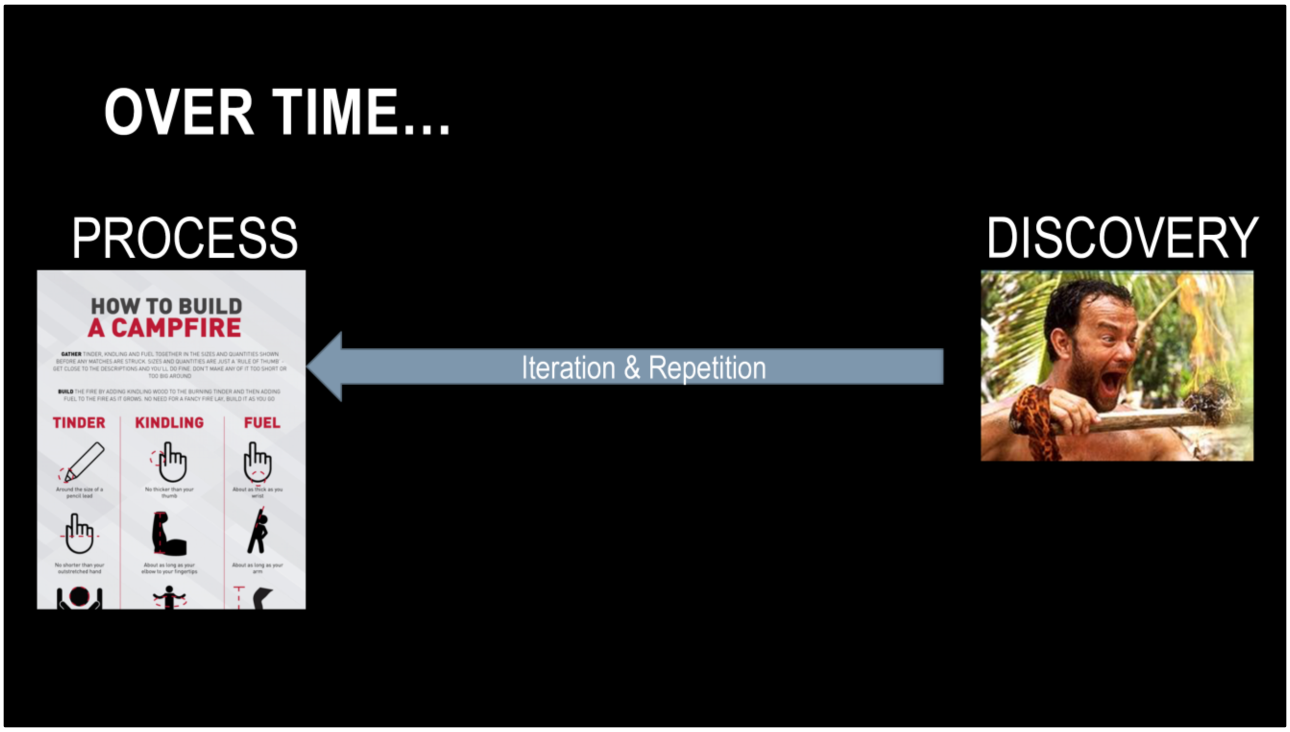
A STORY…

Because they say to always start presentations with stories…

350,000 YEARS AGO

A DISCOVERY…

[Insert tacky Beavis & Butt-Head reference here]



OVER TIME...

PROCESS

HOW TO BUILD
A CAMPFIRE

GATHER TINDER, KINDLING AND FUEL TOGETHER IN THE SIZES AND QUANTITIES SHOWN BEFORE ANY MATCHES ARE STRUCK. SIZES AND QUANTITIES ARE JUST A 'RULE OF THUMB' - GET CLOSE TO THE DESCRIPTIONS AND YOU'LL DO FINE. DON'T MAKE ANY OF IT TOO SHORT OR TOO BIG AROUND

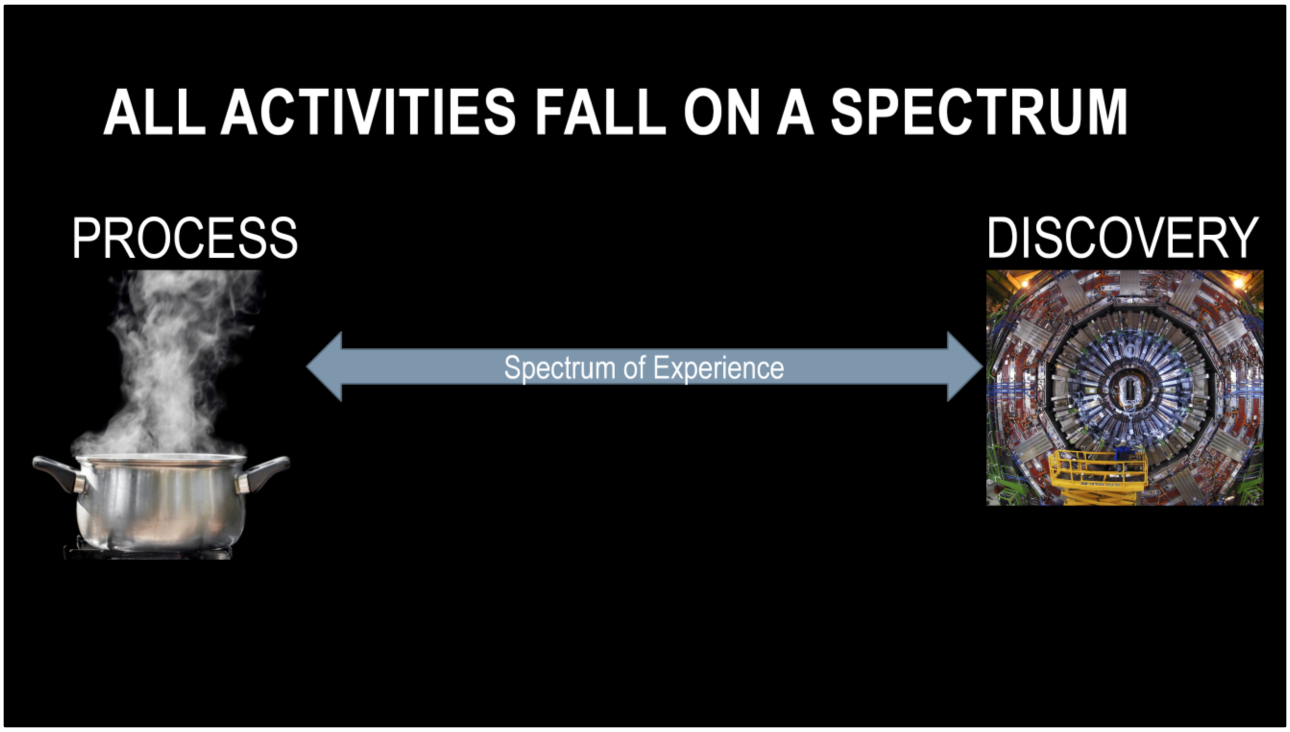
BUILD THE FIRE BY ADDING KINDLING WOOD TO THE BURNING TINDER AND THEN ADDING FUEL TO THE FIRE AS IT GROWS. NO NEED FOR A FANCY FIRE LAY, BUILD IT AS YOU GO

TINDER	KINDLING	FUEL
Around the size of a pencil lead	No thicker than your thumb	About as thick as you wrist
No shorter than your outstretched hand	About as long as your elbow to your fingertips	About as long as your arm

Iteration & Repetition

DISCOVERY

What was once an exciting discovery, over time, iteration, and repetition, becomes something rote and predictable.



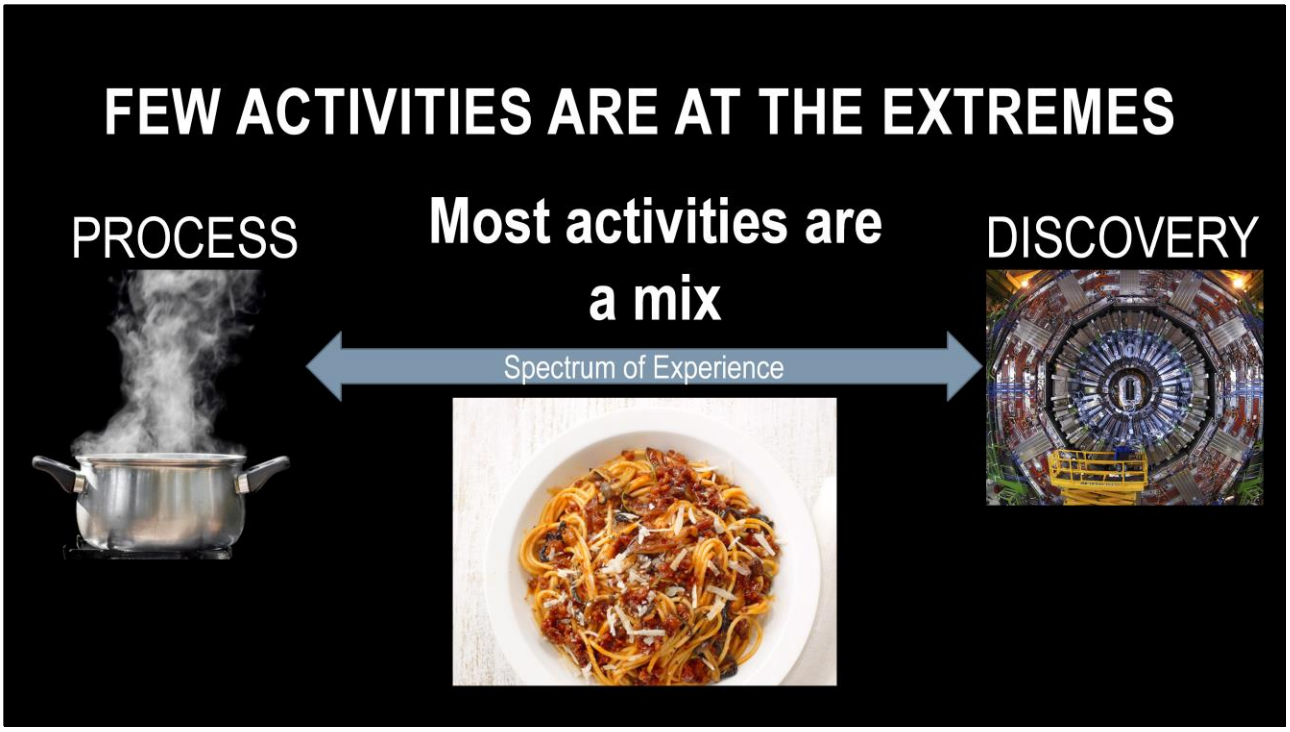
ALL ACTIVITIES FALL ON A SPECTRUM

PROCESS

DISCOVERY

Spectrum of Experience

There is no mystery to boiling water – it's pure process. On the other hand, there are lots of discoveries to be made at the cutting edge of physics!



FEW ACTIVITIES ARE AT THE EXTREMES

PROCESS

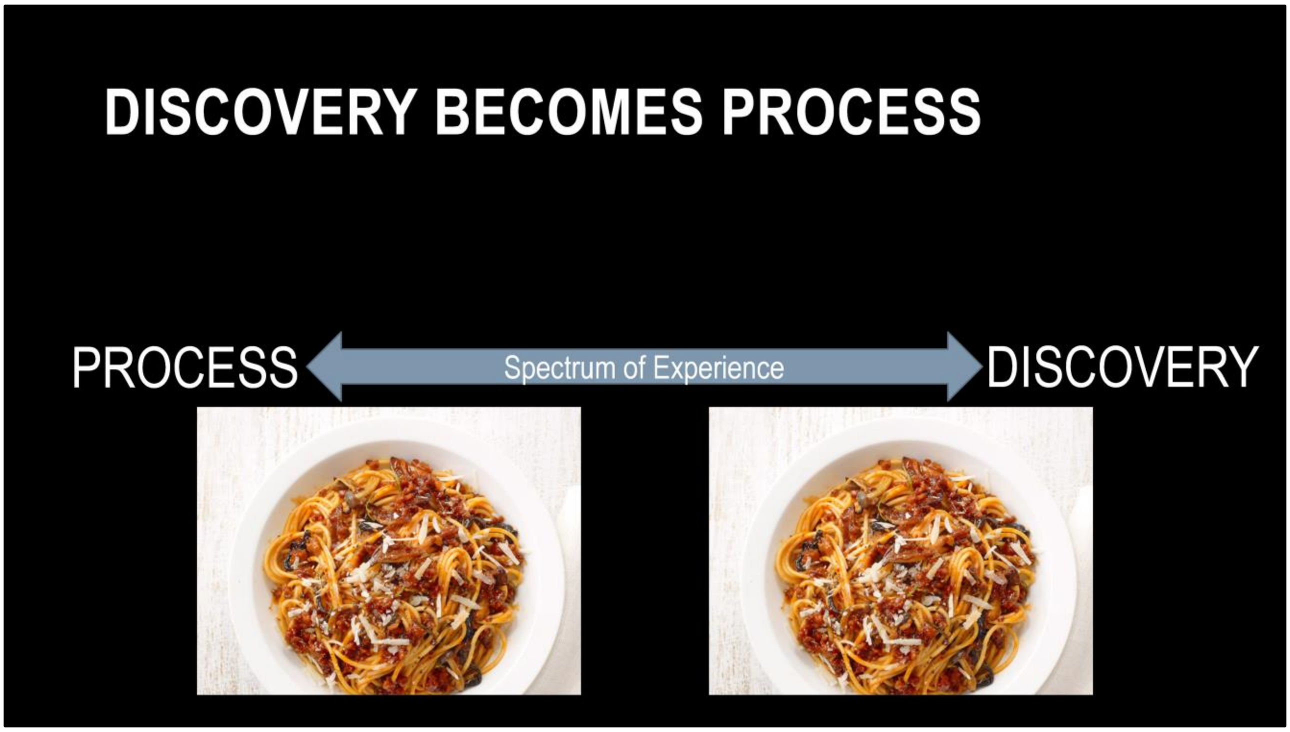
Most activities are a mix

DISCOVERY

Spectrum of Experience

Few activities are pure process (devoid of any experimentation or discovery) and even fewer are pure discovery (totally new and based on no prior knowledge). Even cutting edge physics experiments are built on a foundation of established science.

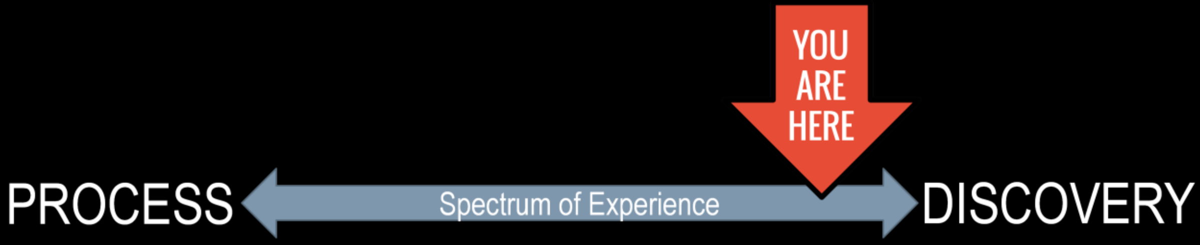
Cooking is a great example of an activity that involves both process and discovery.



DISCOVERY BECOMES PROCESS

As with fire, all activities start on the discovery end of the spectrum and move towards the process end. This applies both globally (somebody had to have been the first person to boil water) and individually. The first time you make your own pasta sauce, there was a lot of discovery. But, by the 1000th time…

OUR BUSINESS IS DISCOVERY



“FIND THE FUN”

"AYE, THERE'S THE RUB"

RISK

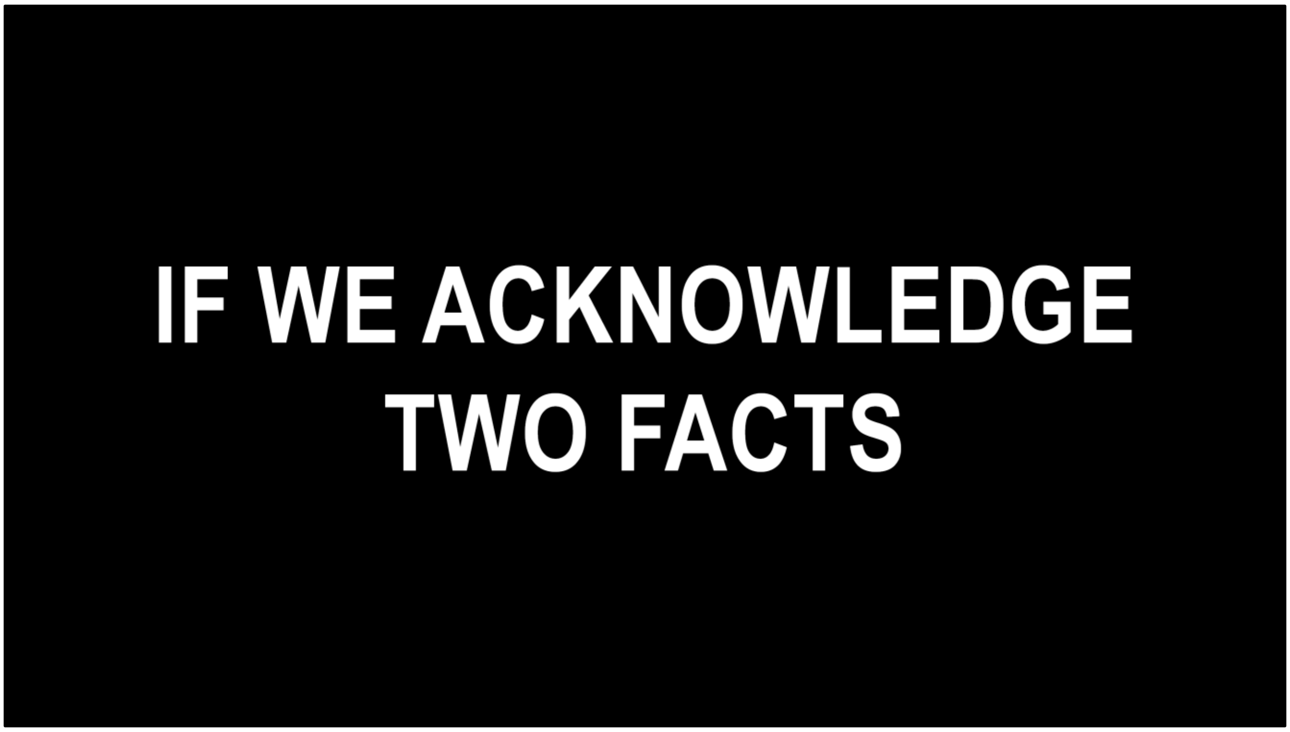
Discovery, by definition, involves the unknown. And the unknown brings with it risk.

VARIANCE

Or as data and operation scientists call it, “variance.” From the perspective of operation science, statistics, finance, etc, the terms “risk” and “variance” are essentially interchangeable.

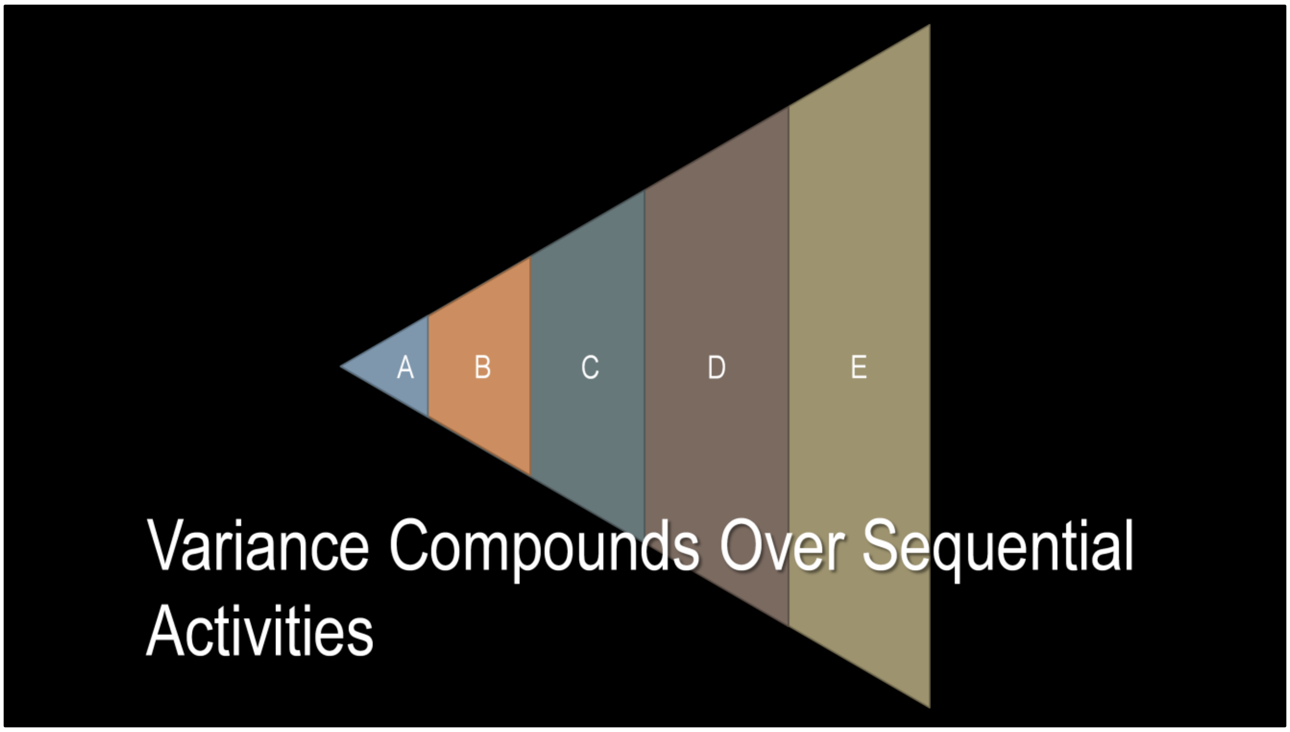
And that variance is the source of so much of our pain when we try to manage long term projects and forecast development.

We can’t just abandon discovery! It’s what makes our jobs fun!



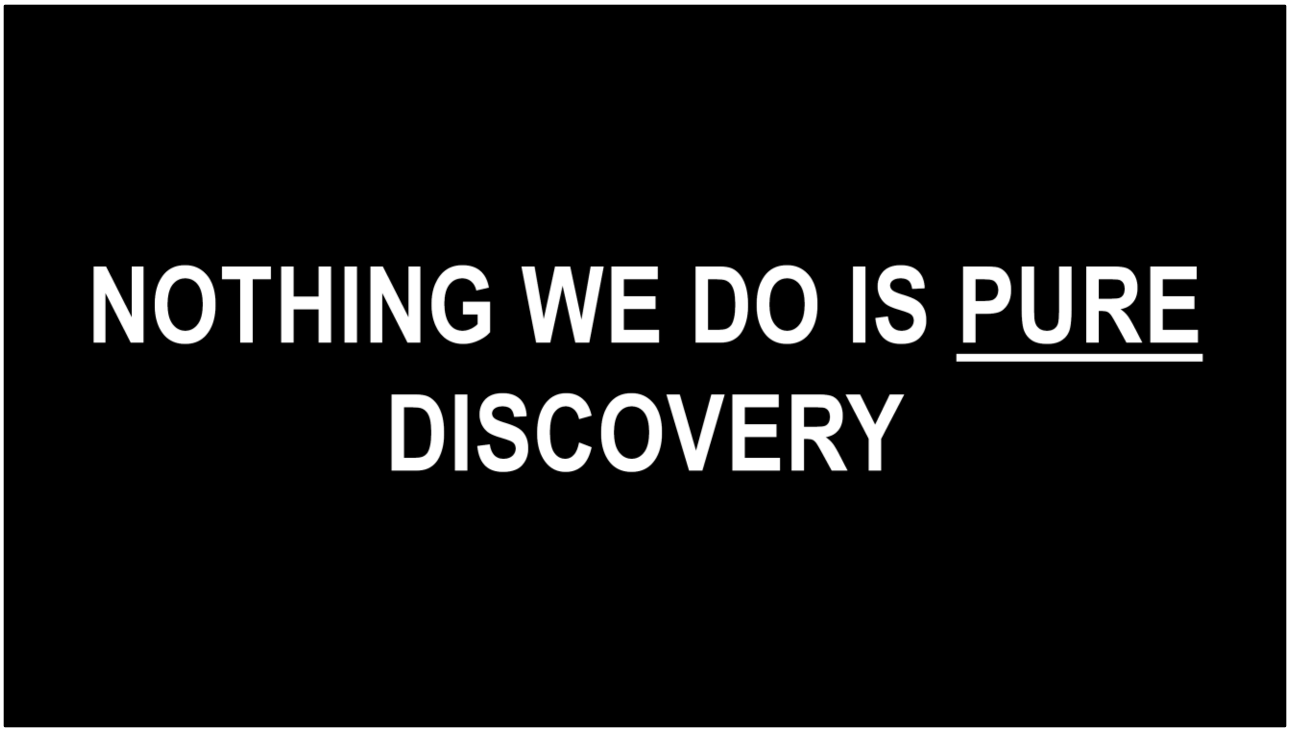
IF WE ACKNOWLEDGE TWO FACTS

We don’t need to resign ourselves to the ravages of variance



Variance Compounds Over Sequential Activities

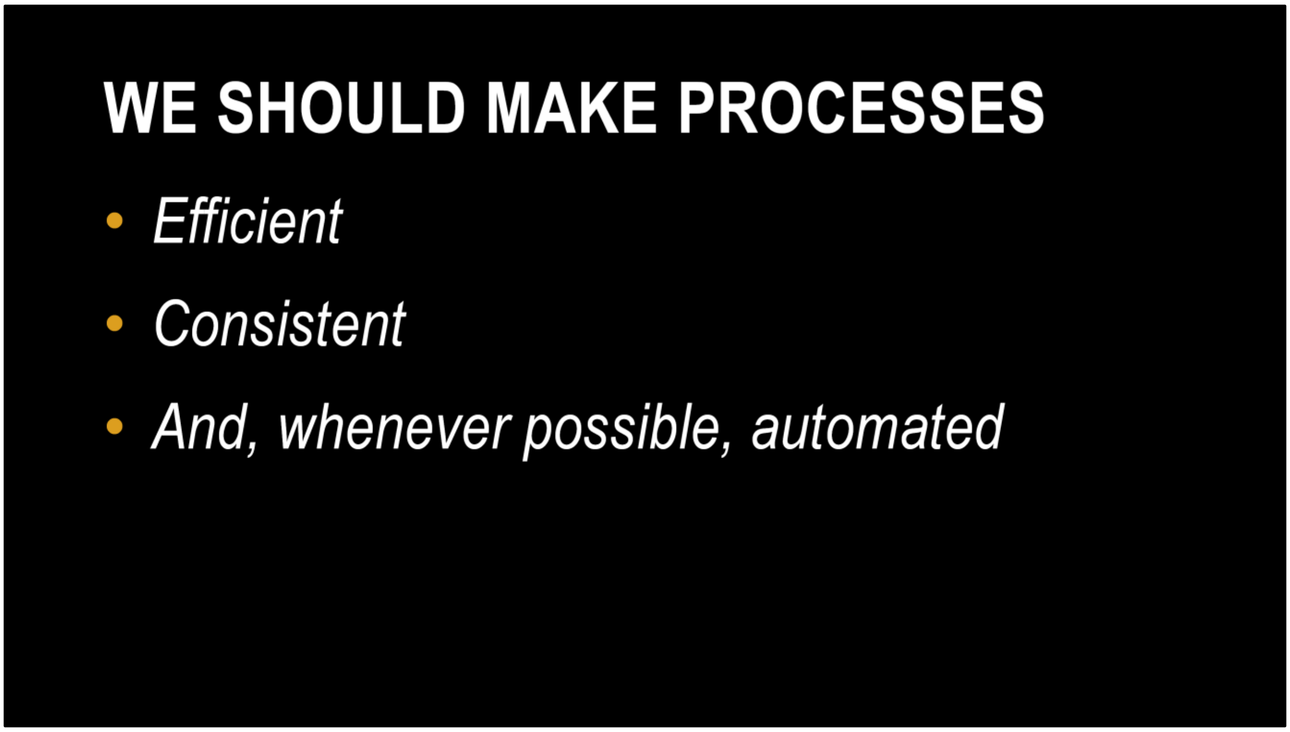
If every activity, A-E has a distribution of outcomes of 1 throgh 5, then the total distribution of outcomes to get from the start of A to the end of E is 5 through 25.



NOTHING WE DO IS PURE DISCOVERY

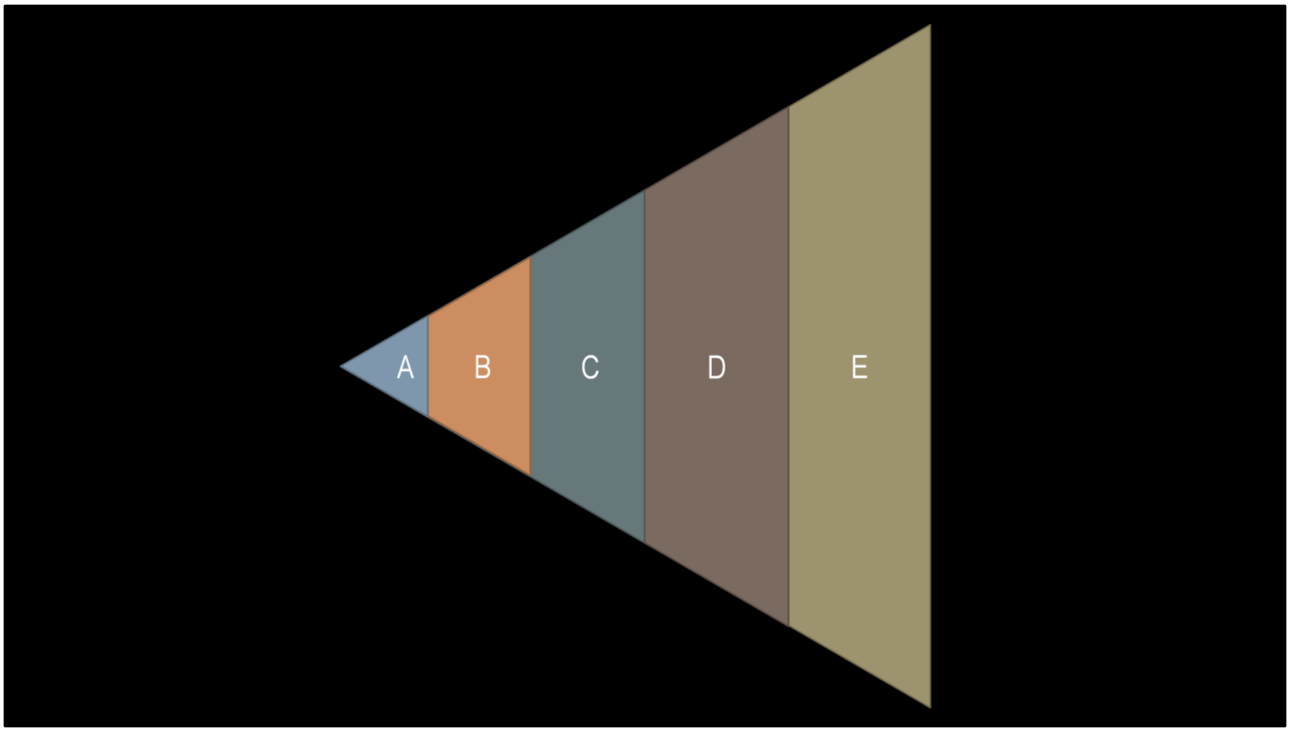
Even the most avant garde, experimental, in the weeds, exploratory design is still supported by some degree of process (spec’ing features, coding them, building them, running QA passes)

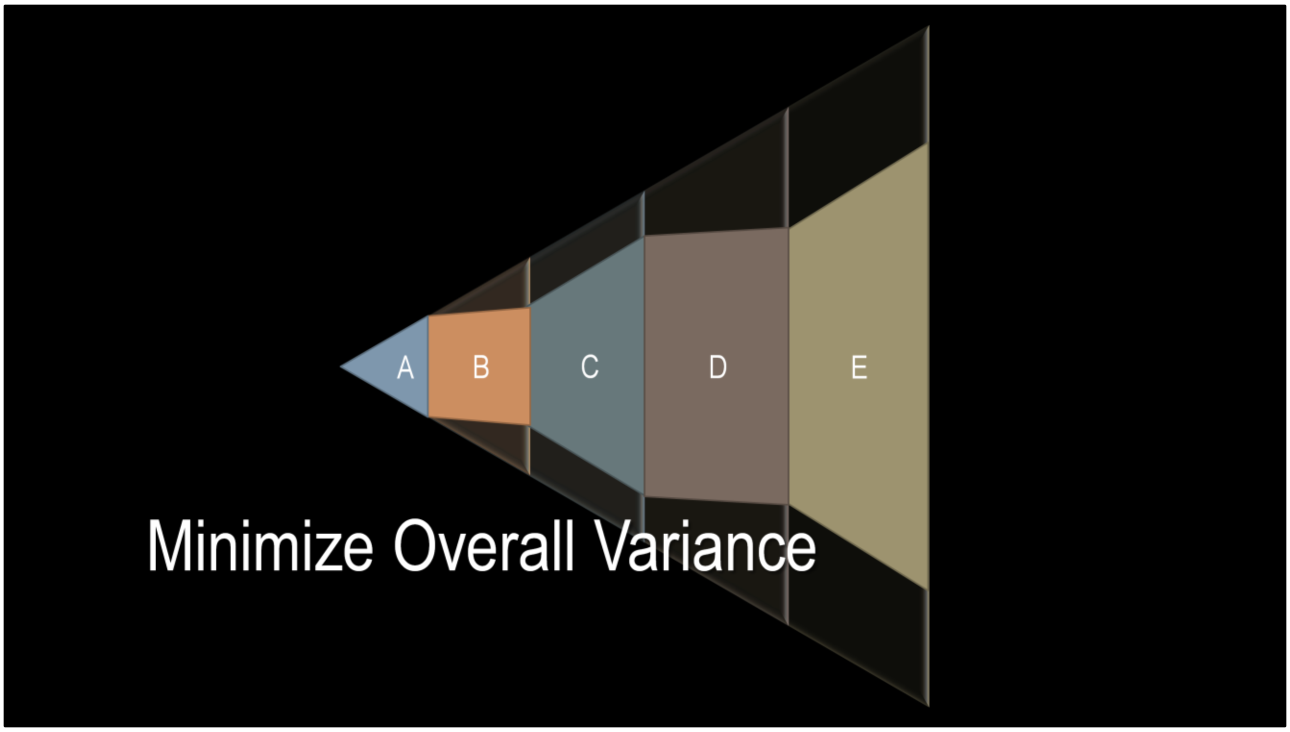
That’s why we need all this stuff!



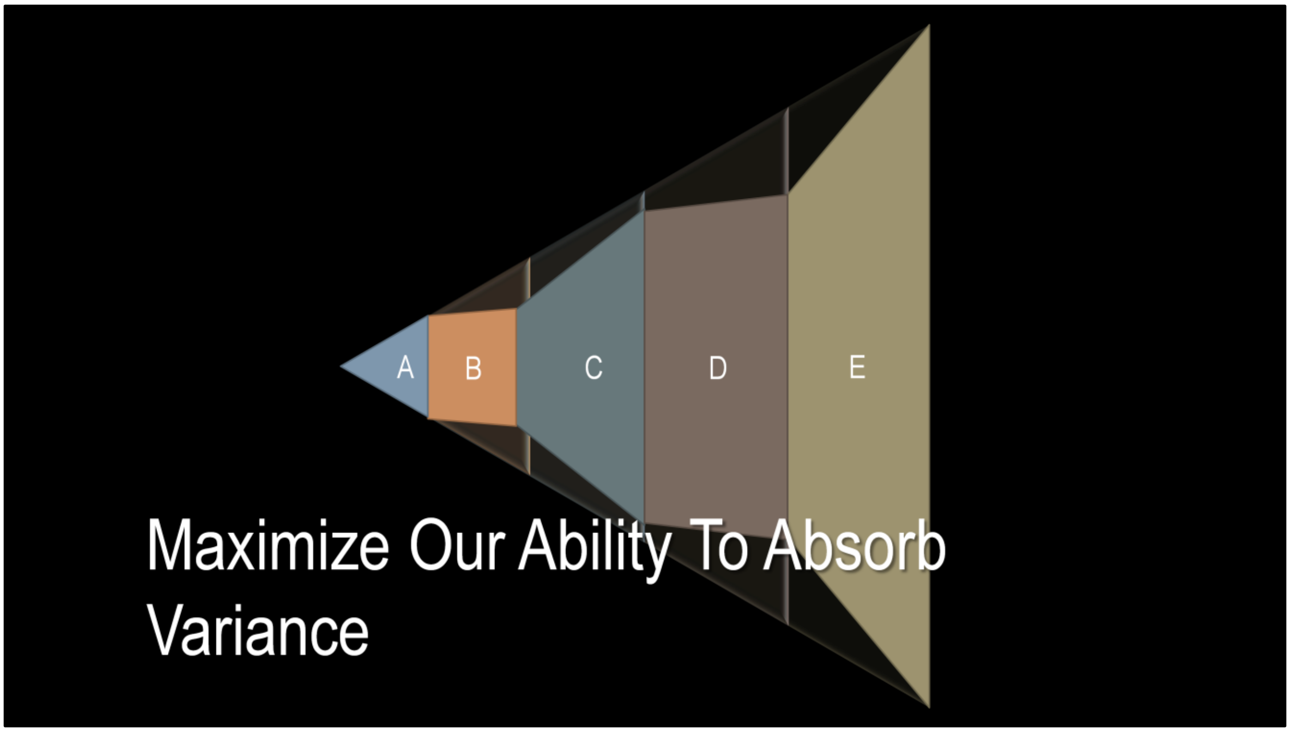
WE SHOULD MAKE PROCESSES

- *Efficient*
- *Consistent*
- *And, whenever possible, automated*


A
B
C
D
E



EG, if we can shrink the outcome distributions of B and D from 1-5 down to a known outcome of 1, the distribution total outcomes shrinks from 5-25 to 5-17

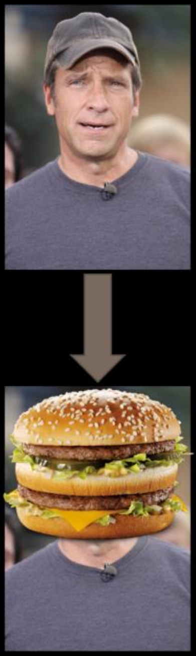


Maximize Our Ability To Absorb Variance

EG, if we shrink narrow the possible outcomes of B and D to just 1, then the combined variance of A, C, and E could increase by 8 without changing the overall outcome distribution of 5-25.

ROADMAP

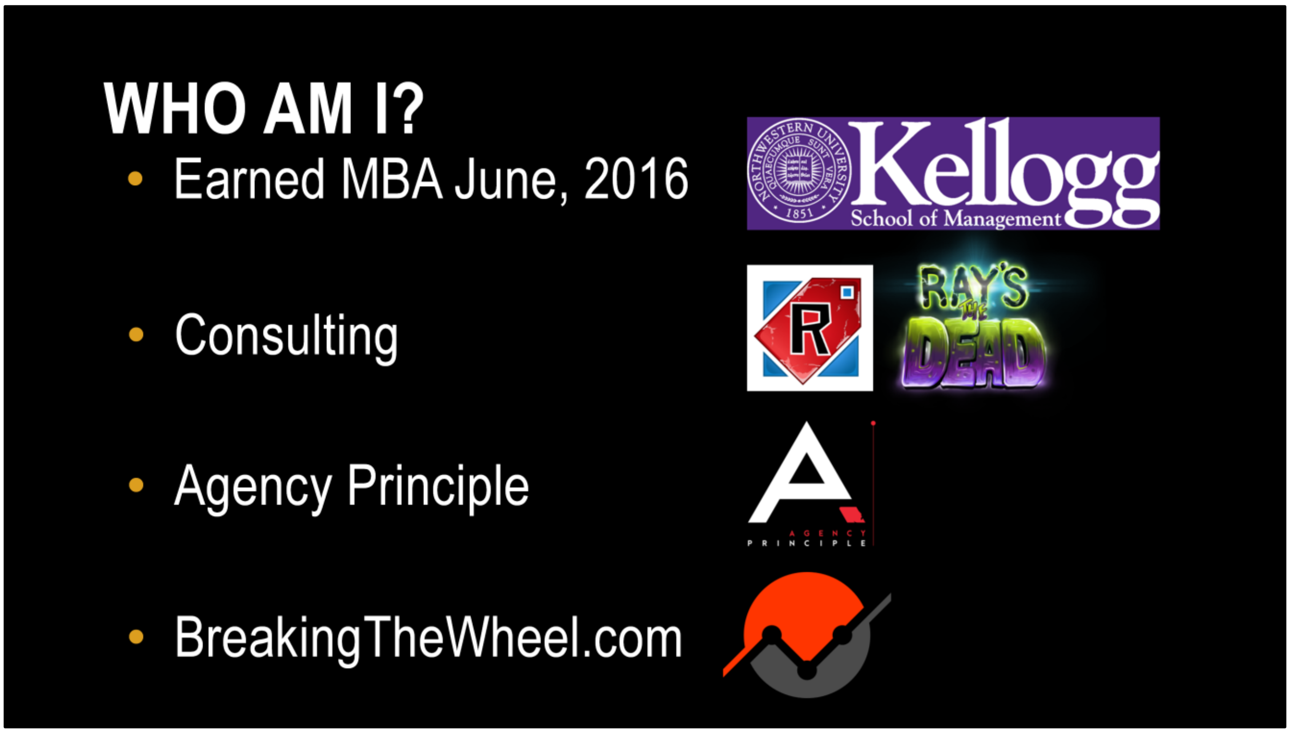
- Quick Bio
- Process Flow Fundamentals
- Capacity Charts
- Lean Production for Games
- Closing Thoughts
- Q&A



WHO AM I?

- Started in the industry in Jan 2007

WIDELOAD



WHO AM I?

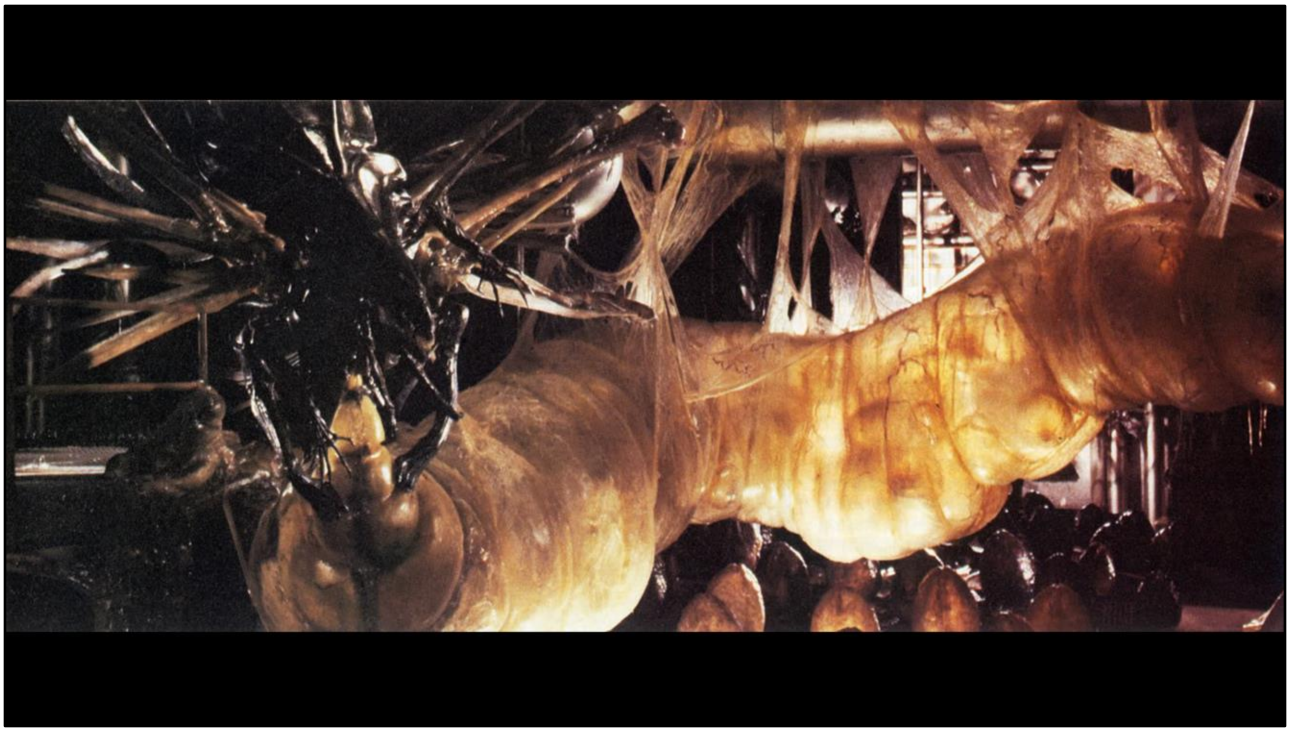
- Earned MBA June, 2016
- Consulting
- Agency Principle
- BreakingTheWheel.com

MY FAVORITE MOVIE:

ALIENS

Maybe it’s cliché, but Aliens is my favorite movie ever.

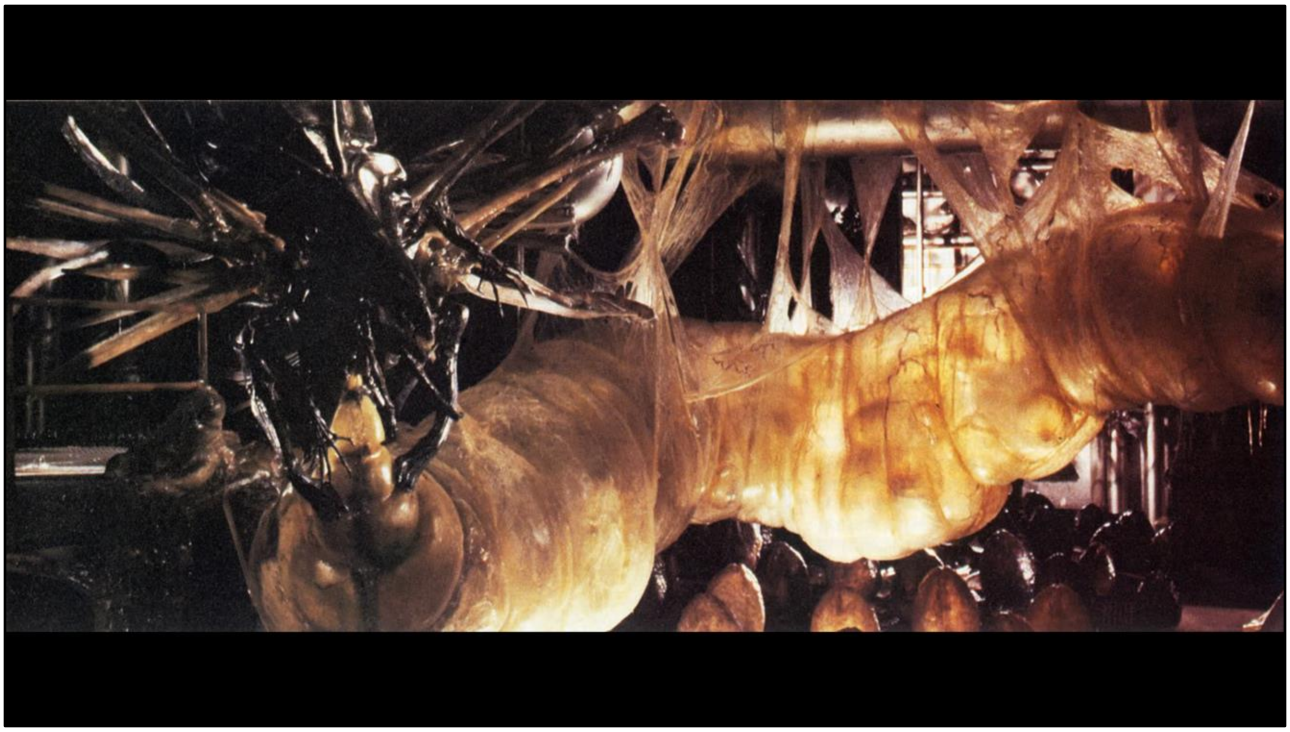
At the climax of the movie, Ripley and Newt stumble right into the heart of the nest and meet the Queen



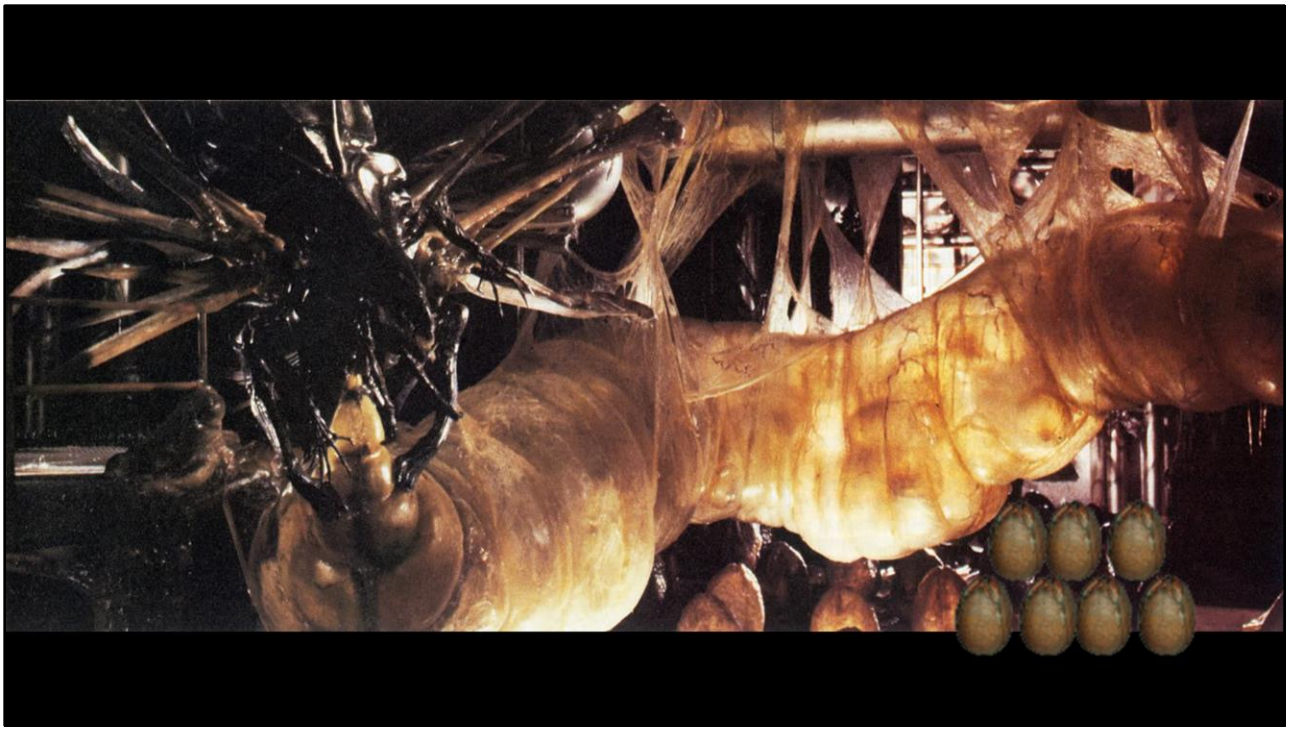
And the Queen has this gross, slimy, bulbous sack through which she deposits face-hugger eggs on the hive floor

HOW MANY EGGS DOES SHE HAVE IN THAT THING?

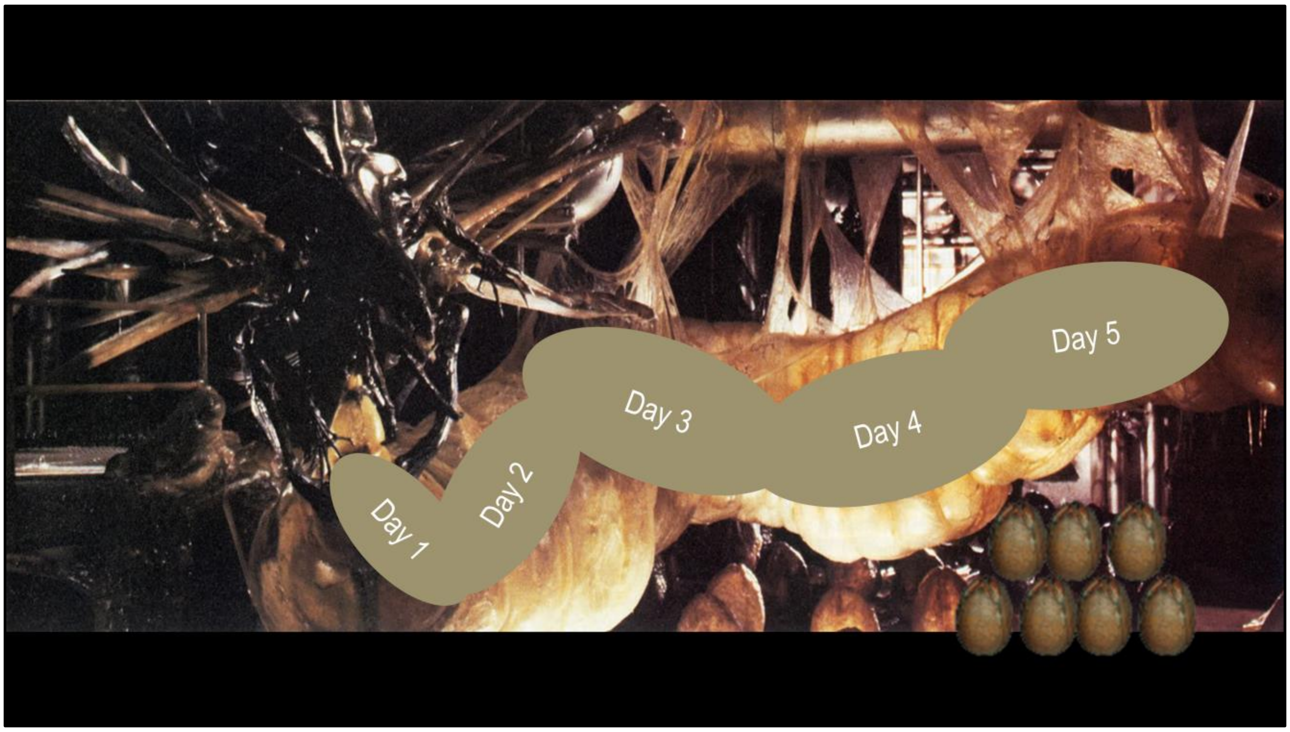
LET’S FIND OUT!



So, here’s our Queen…



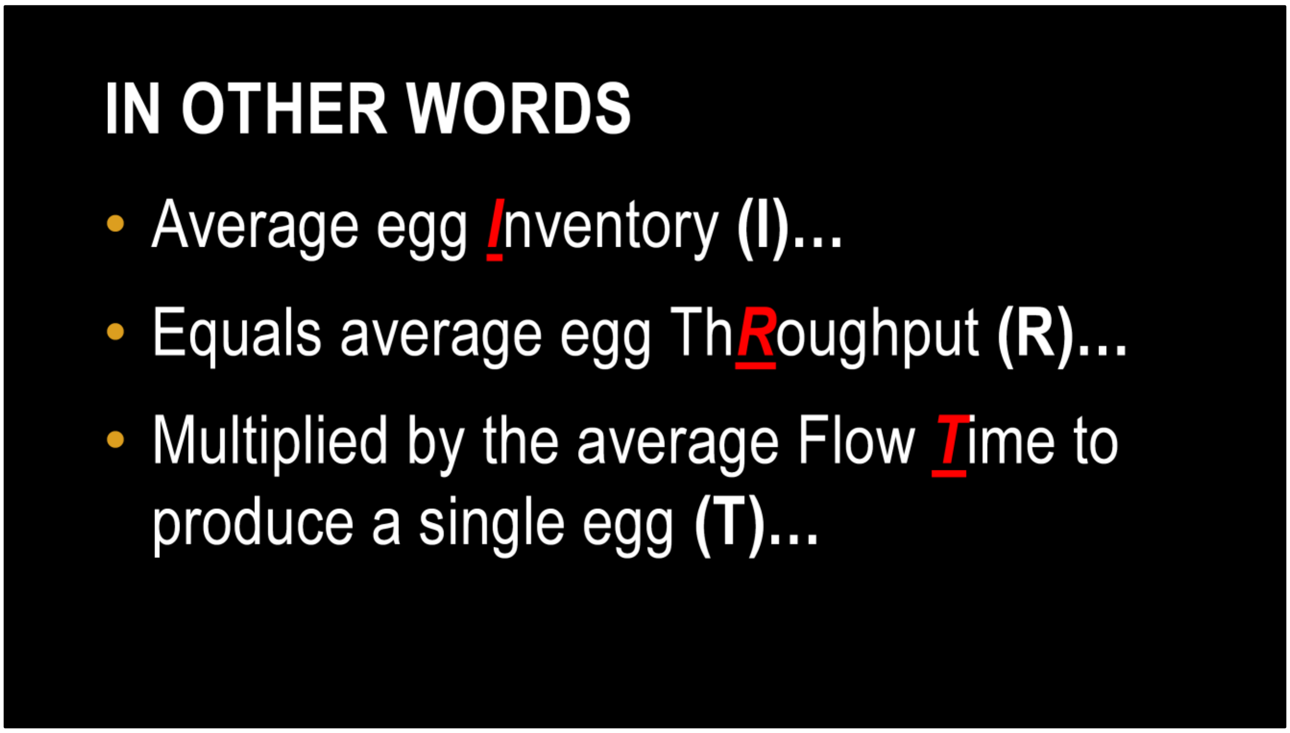
Let’s assume that, on average, the Queen lays 7 face-hugger eggs a day



Let’s also assume it takes the queen 5 days, on average, to gestate a single egg

If that is the case, the only way the Queen can sustain a 7-egg-a-day throughput with a 5-day per egg turnaround time is if there are 7 eggs in each phase of gestation

THEREFORE, ON AVERAGE, THE QUEEN HAS ***7*5 = 35*** EGGS IN THE SACK

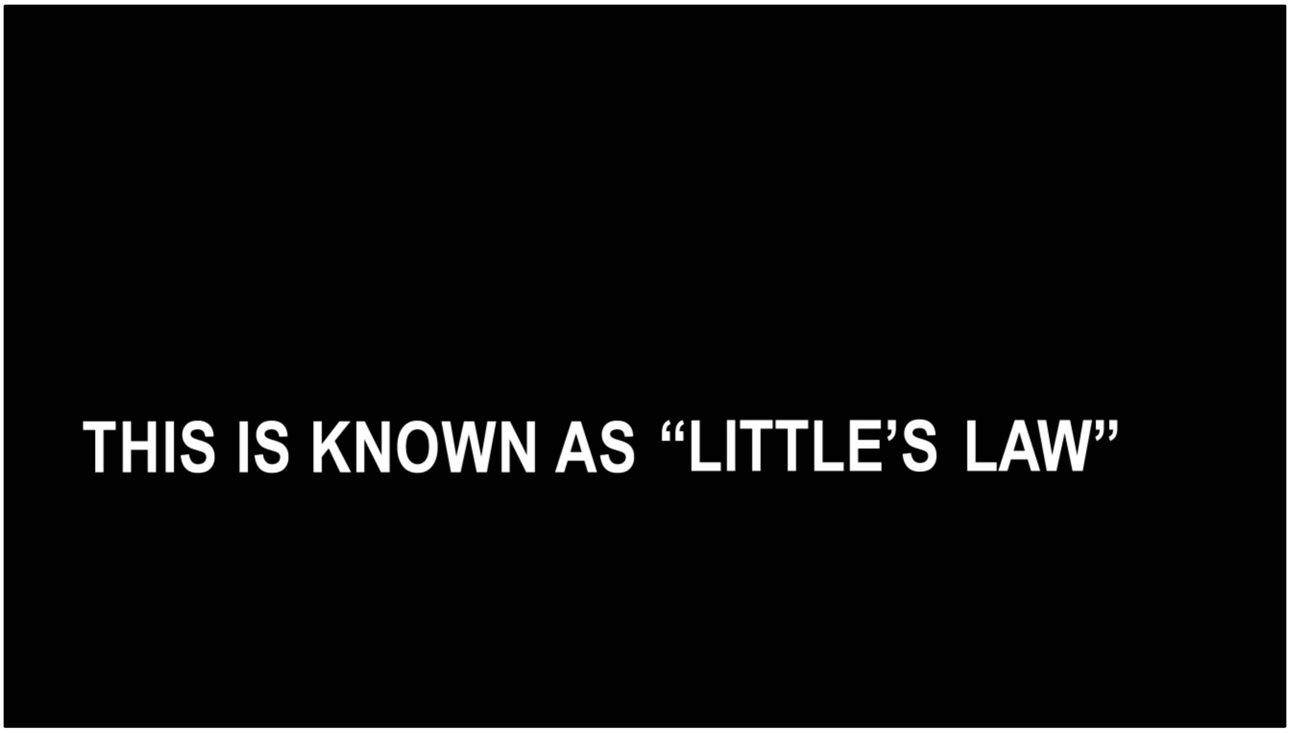


IN OTHER WORDS

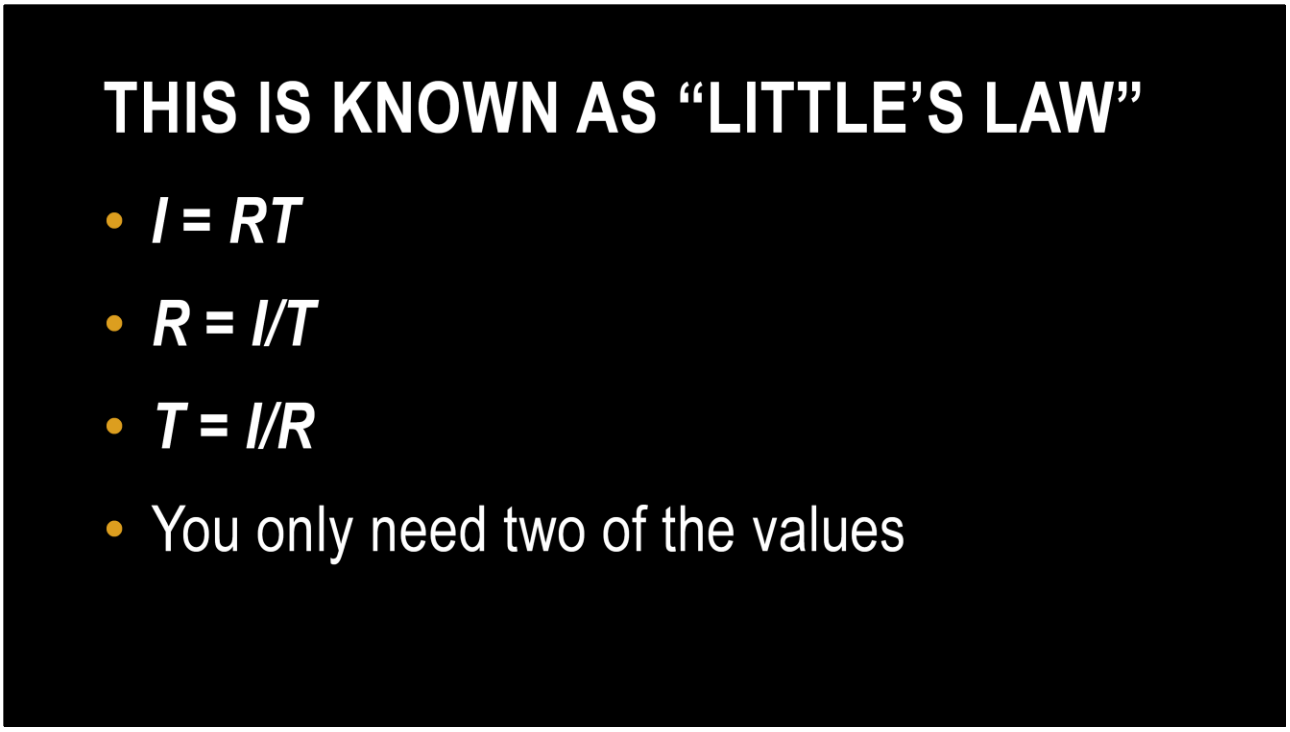
- Average egg ***I***nventory **(I)…**
- Equals average egg Th***R***oughput **(R)…**
- Multiplied by the average Flow ***T***ime to produce a single egg **(T)…**

OR, MORE SIMPLY…

$$I = RT$$

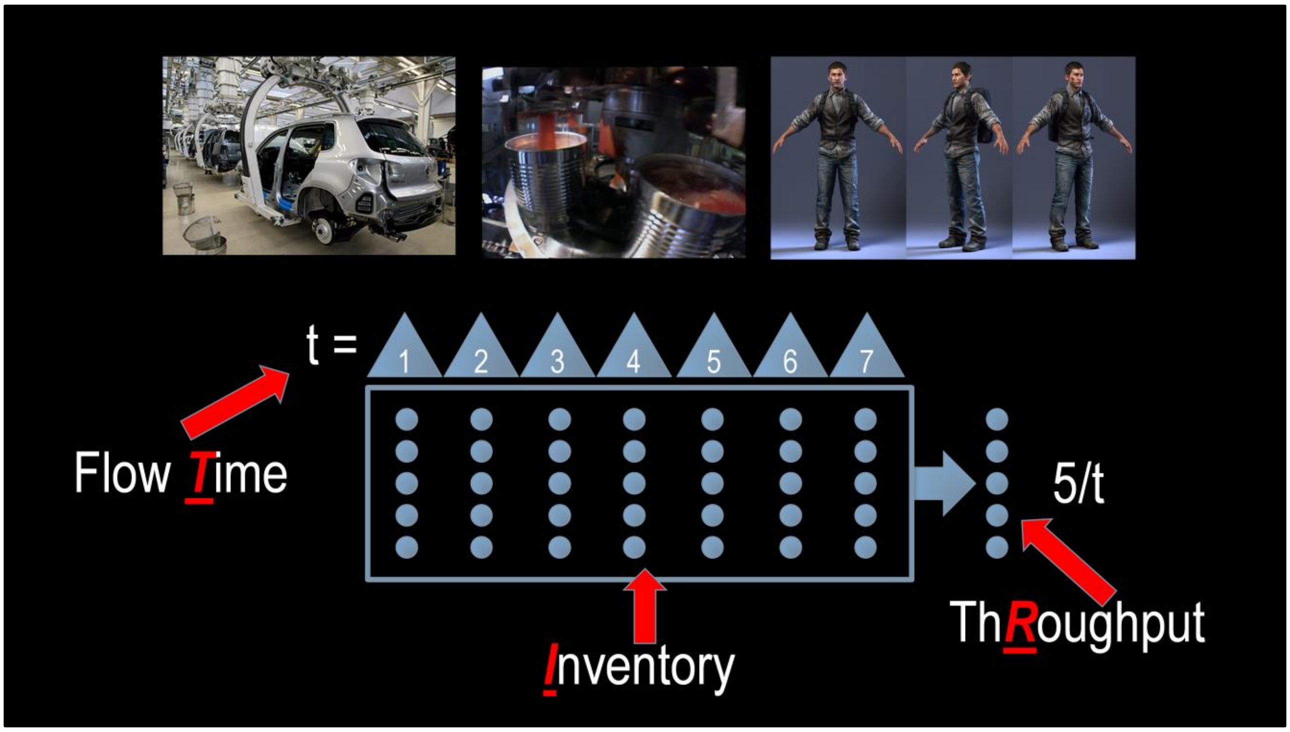


THIS IS KNOWN AS "LITTLE'S LAW"



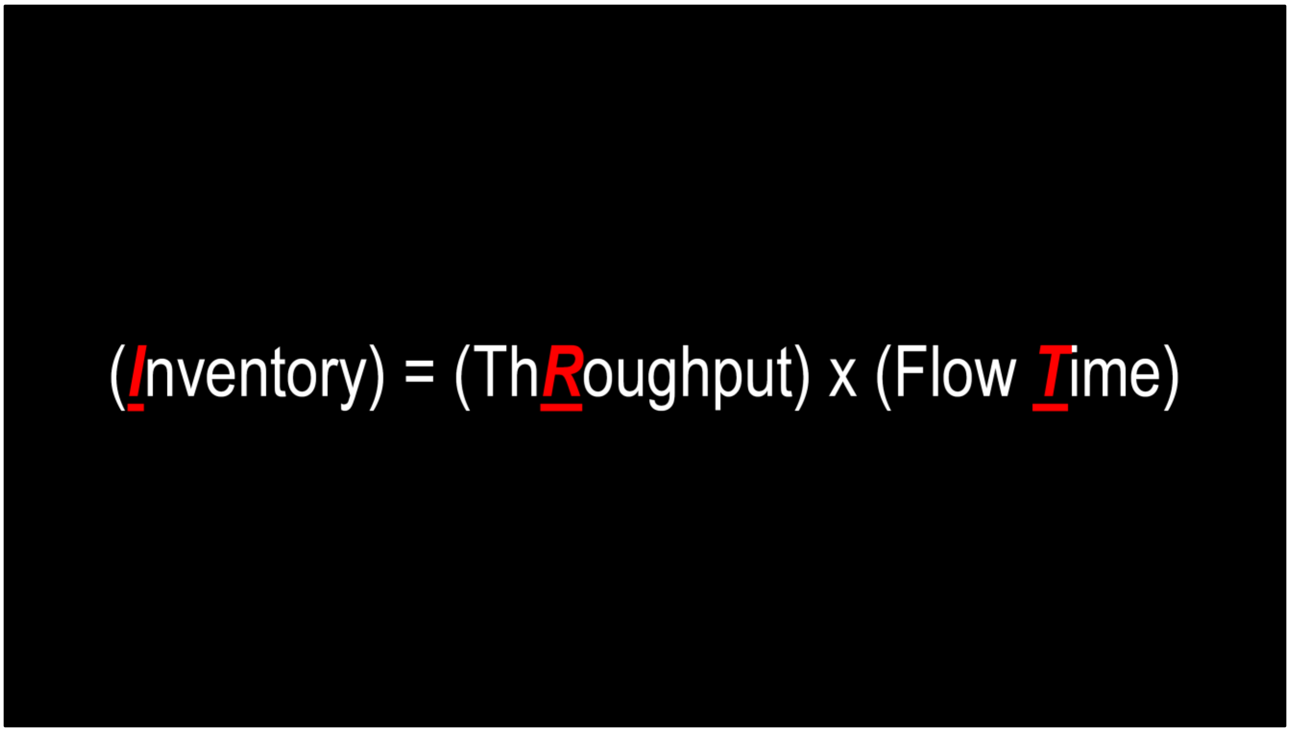
THIS IS KNOWN AS “LITTLE’S LAW”

- $I = RT$
- $R = I/T$
- $T = I/R$
- You only need two of the values



(Belaboring the point to make sure it’s clear)
For any ongoing process (assembling cars, manufacturing cans of soup, or generating game assets):

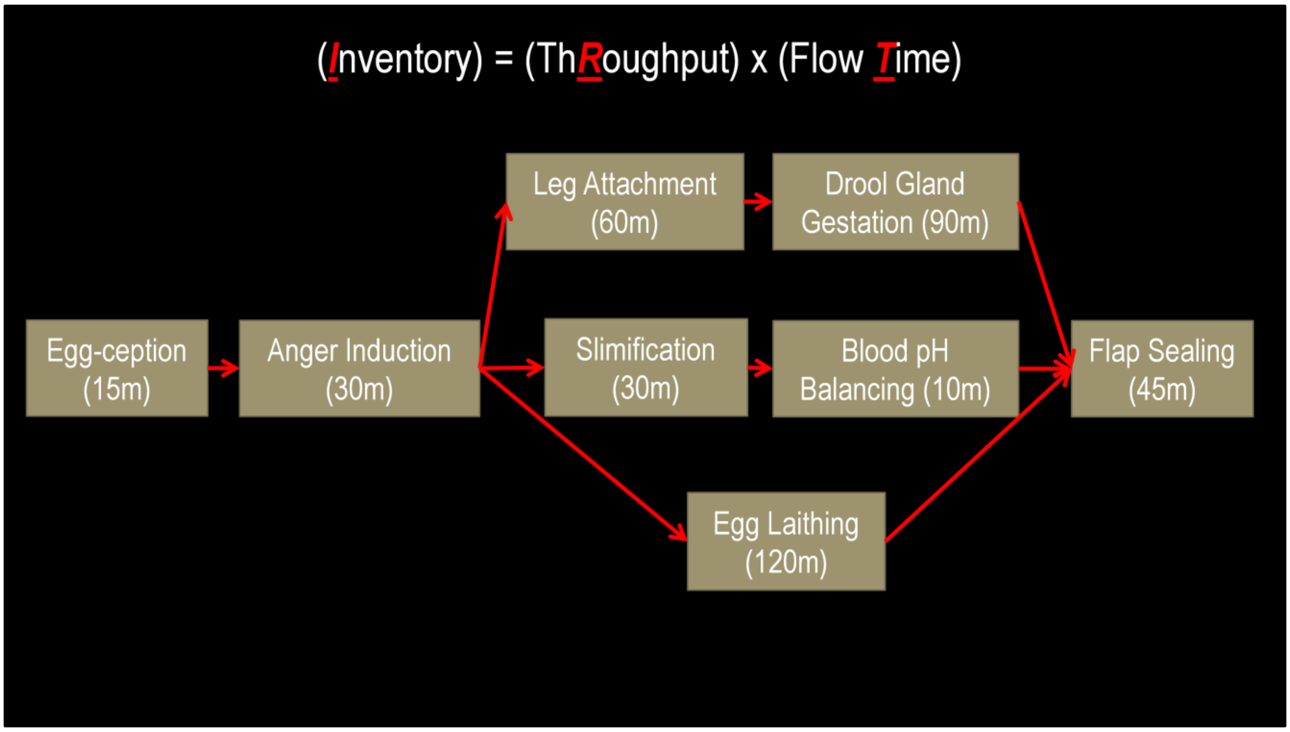
- The average amount of things currently being processed (the inventory)…
- …is equal to the average rate at which things come out of the process (the throughput)…
- …multiplied by the average time to process a single thing (the flow time)



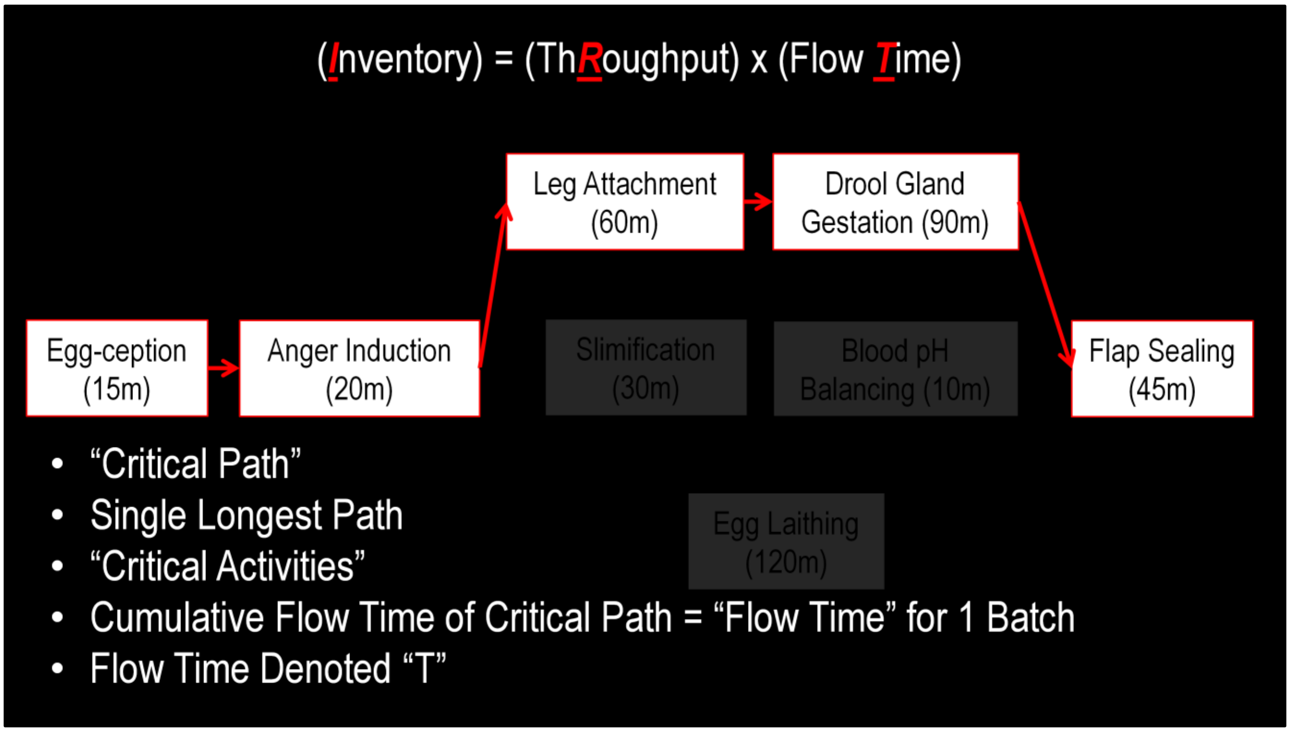
(*I*nventory) = (Th*R*oughput) x (Flow *T*ime)

I will be brining this equation up contextually throughout the slides

HOW CAN YOU DETERMINE THOSE VALUES?



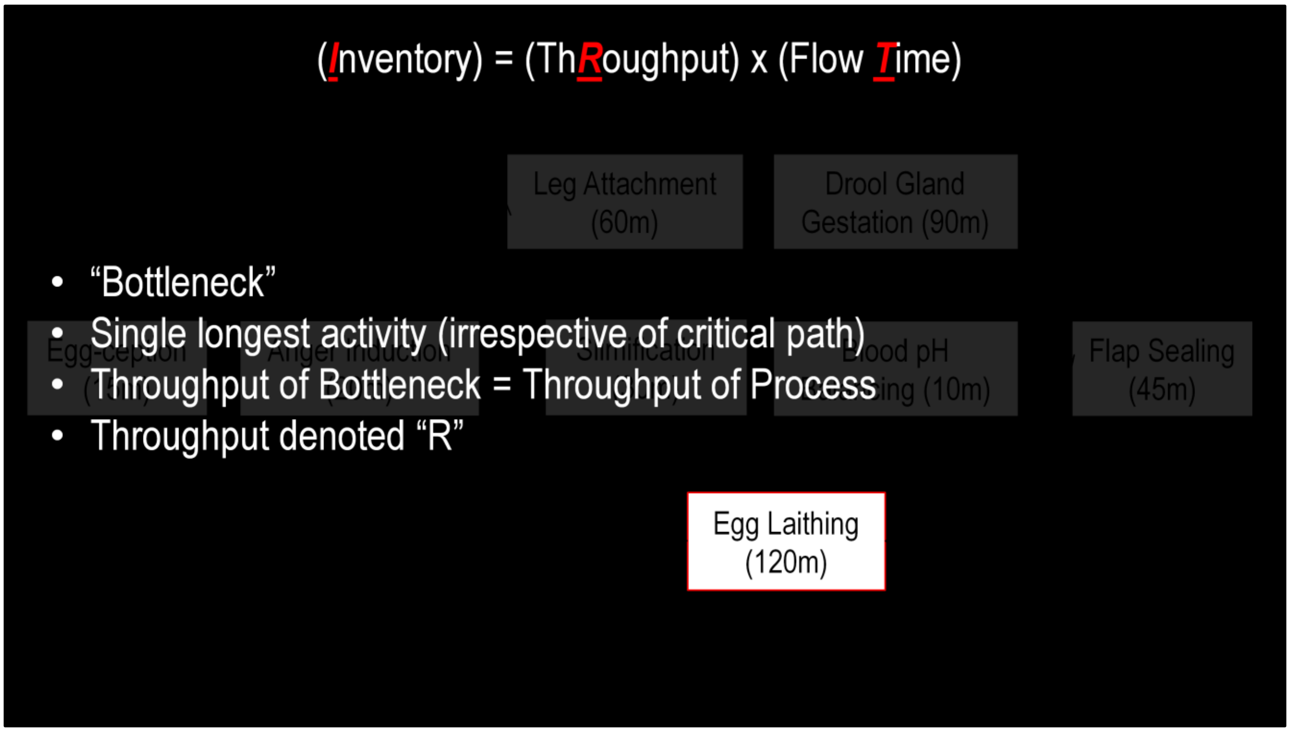
Imagine that inside the egg sack is a complex sequence of 'activities' that are all necessary to generate a complete face hugger egg



(*I*nventory) = (Th*R*oughput) x (Flow *T*ime)

- "Critical Path"
- Single Longest Path
- "Critical Activities"
- Cumulative Flow Time of Critical Path = "Flow Time" for 1 Batch
- Flow Time Denoted "T"

One path is clearly longer than the others in terms of cumulative time. This is the "critical path".

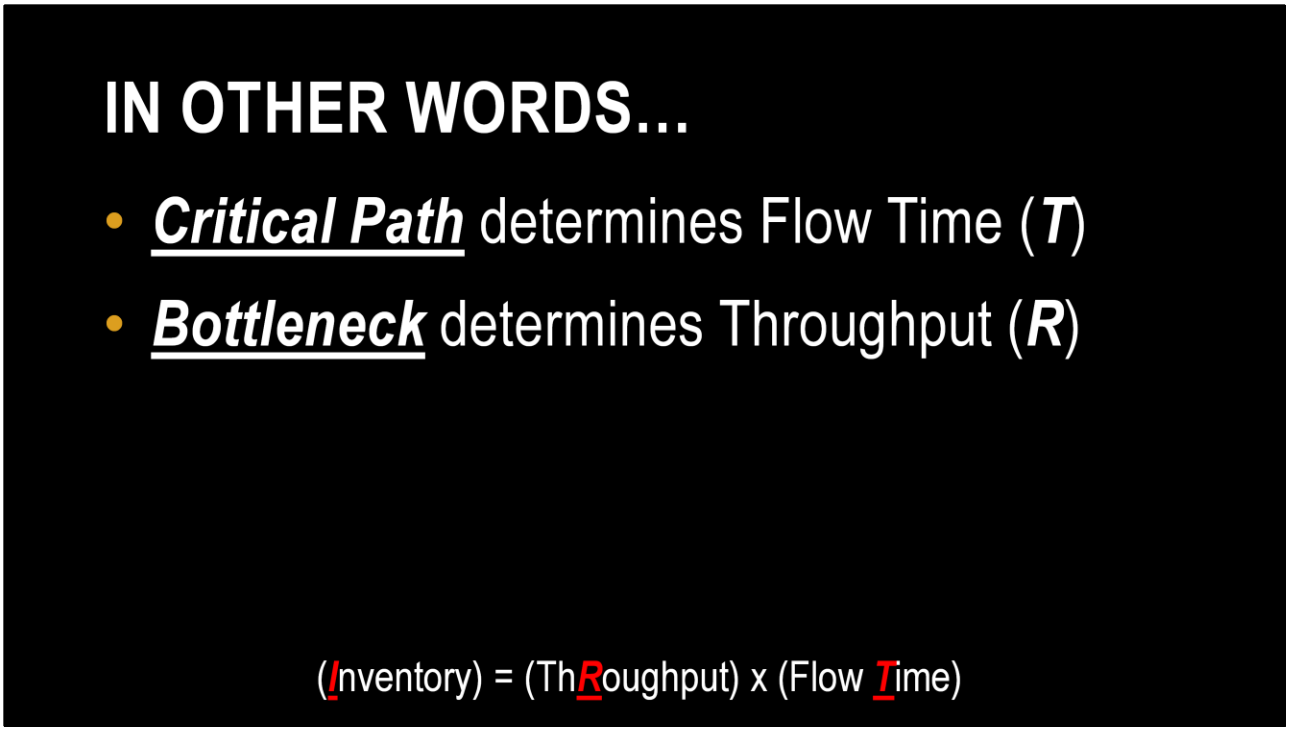


(*I*nventory) = (Th*R*oughput) x (Flow *T*ime)

- "Bottleneck"
- Single longest activity (irrespective of critical path)
- Throughput of Bottleneck = Throughput of Process
- Throughput denoted "R"

Leg Attachment (60m)
Drool Gland Gestation (90m)
Flap Sealing (45m)
Egg Laithing (120m)

Let's also notice that one activity is distinctly longer than the others. This is the "bottleneck."

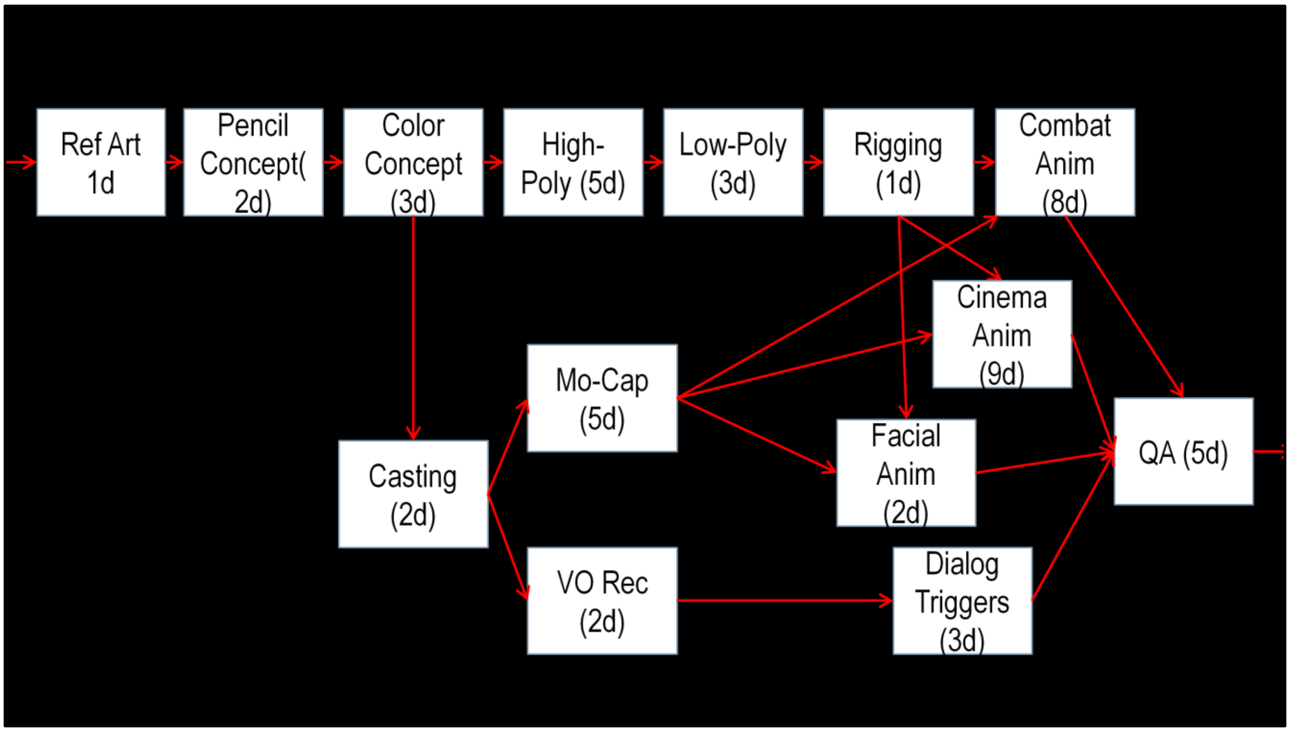


IN OTHER WORDS…

- ***Critical Path*** determines Flow Time (***T***)
- ***Bottleneck*** determines Throughput (***R***)

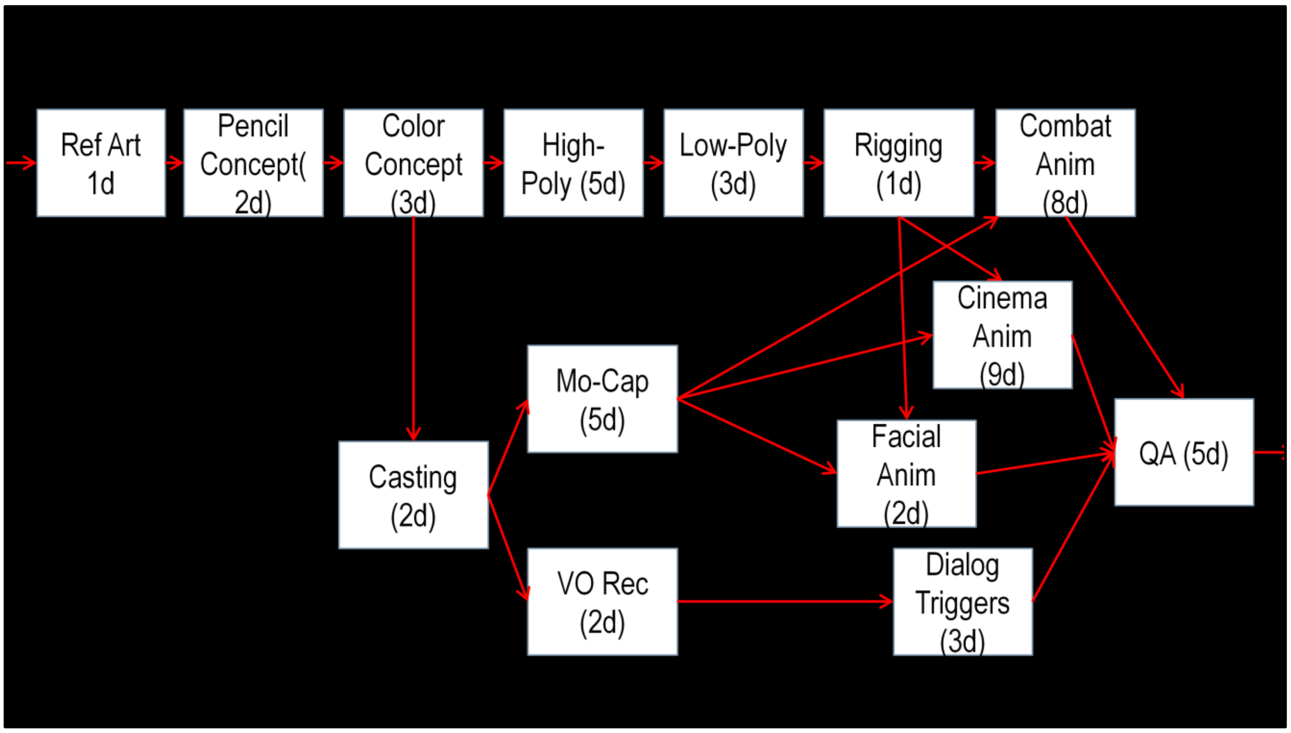
(***I***nventory) = (Th***R***oughput) x (Flow ***T***ime)

A MORE RELEVANT EXAMPLE…

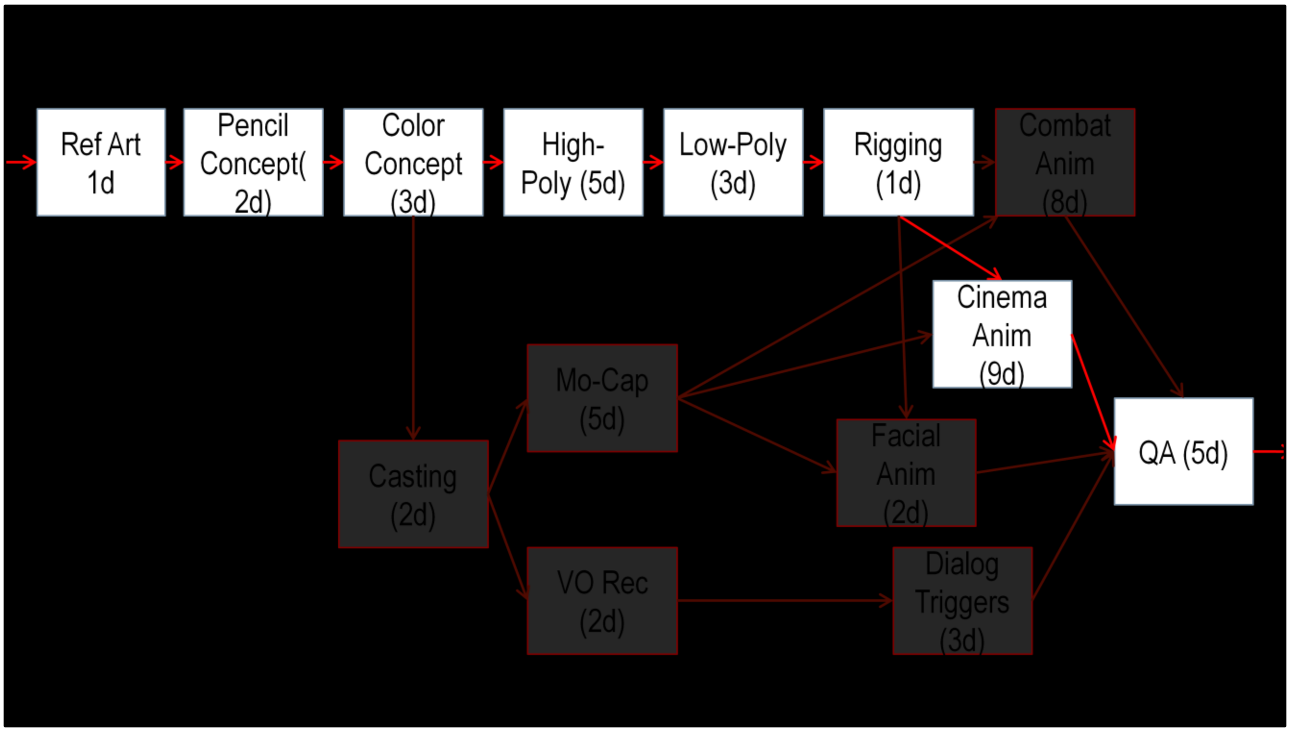


Here is a hypothetical pipeline for character asset creation

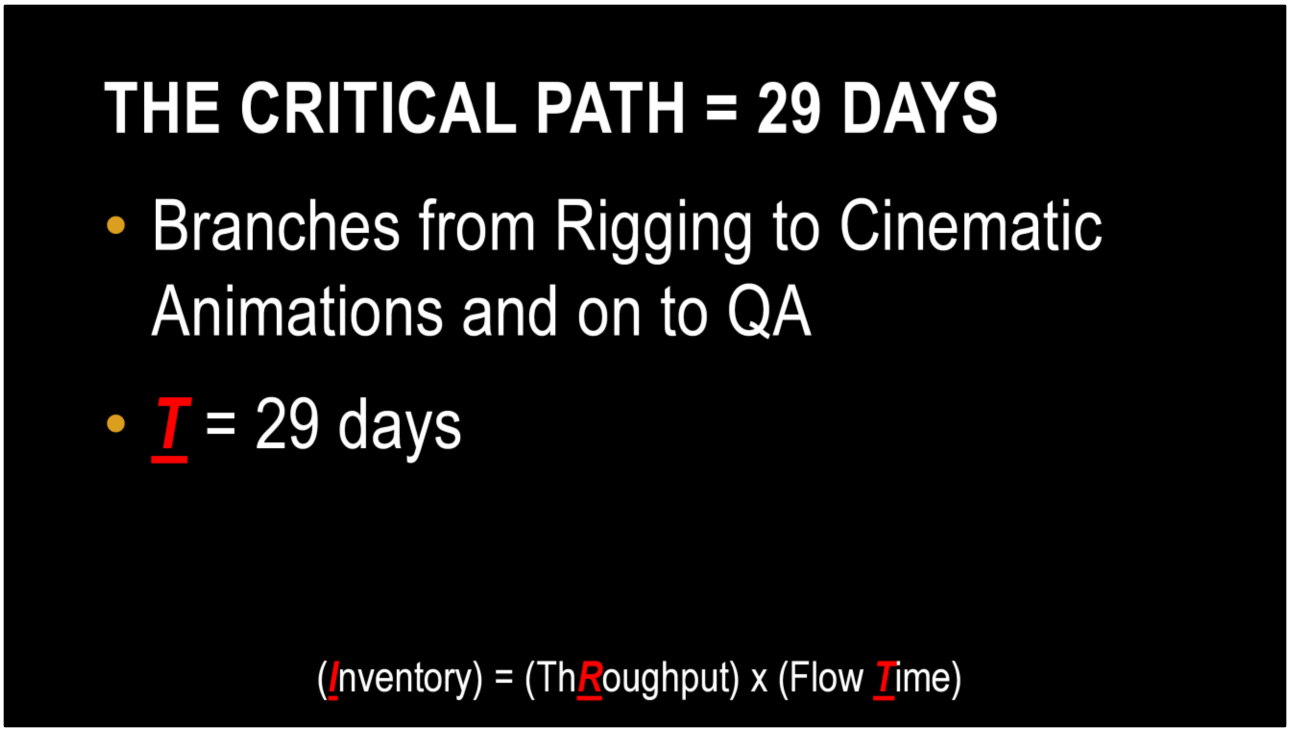
THIS PIPELINE *SEEMS* COMPLEX, BUT OPERATIONS SCIENCE GREATLY SIMPLIFIES THE ANALYSIS



This slide automatically transitions to the next to provide contrast between overall pipeline and critical path



Here is our critical path

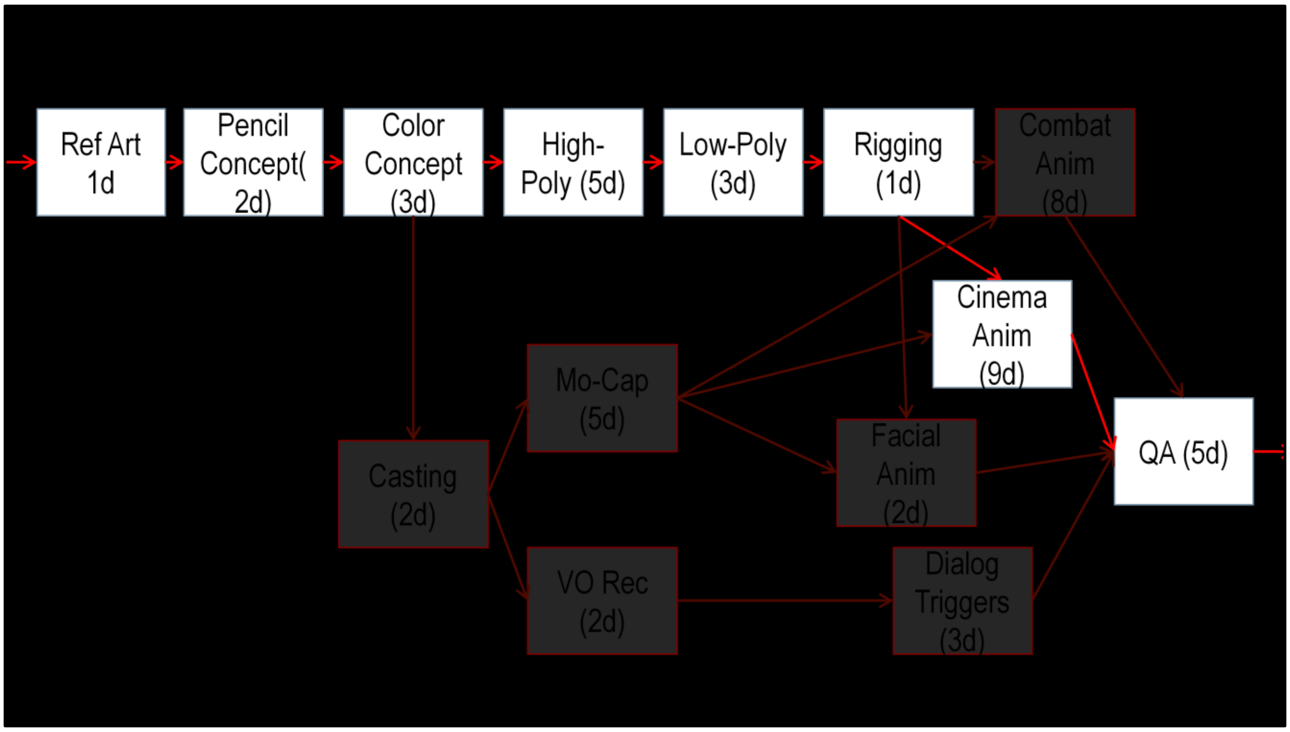


THE CRITICAL PATH = 29 DAYS

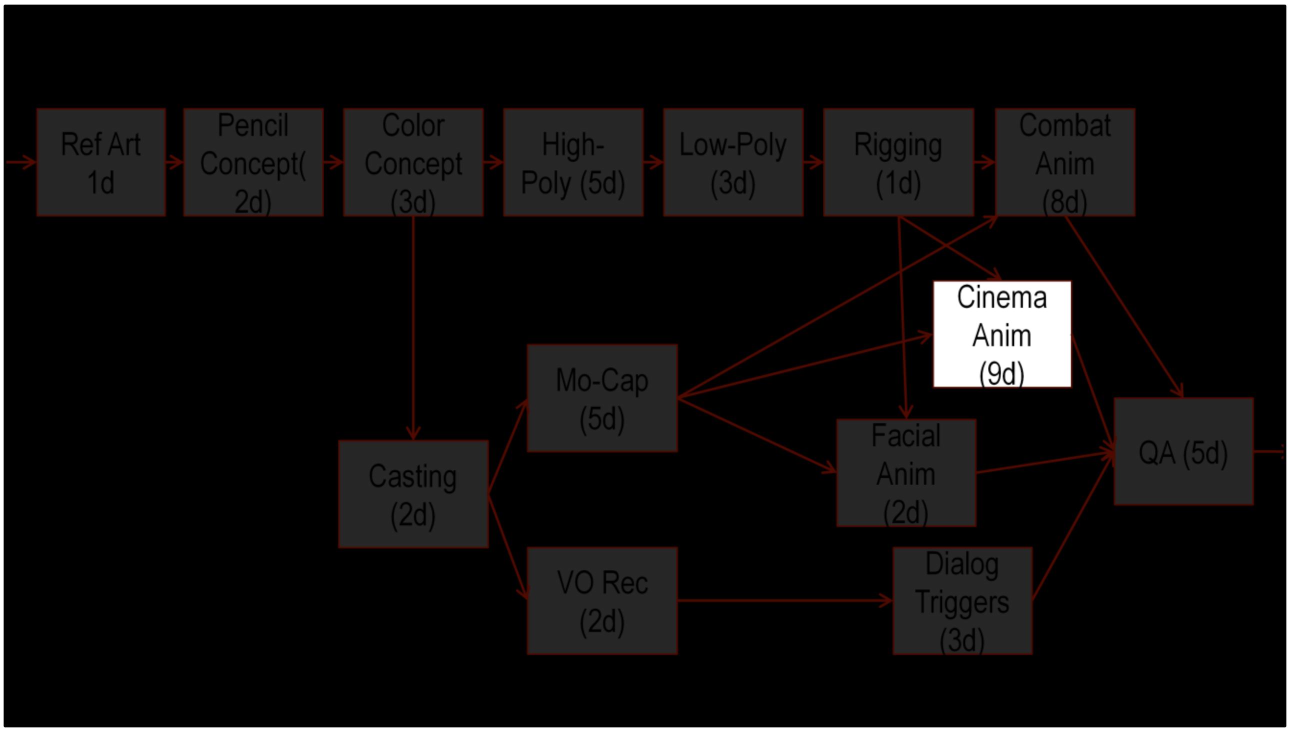
- Branches from Rigging to Cinematic Animations and on to QA
- ***T*** = 29 days

(***I***nventory) = (Th***R***oughput) x (Flow ***T***ime)

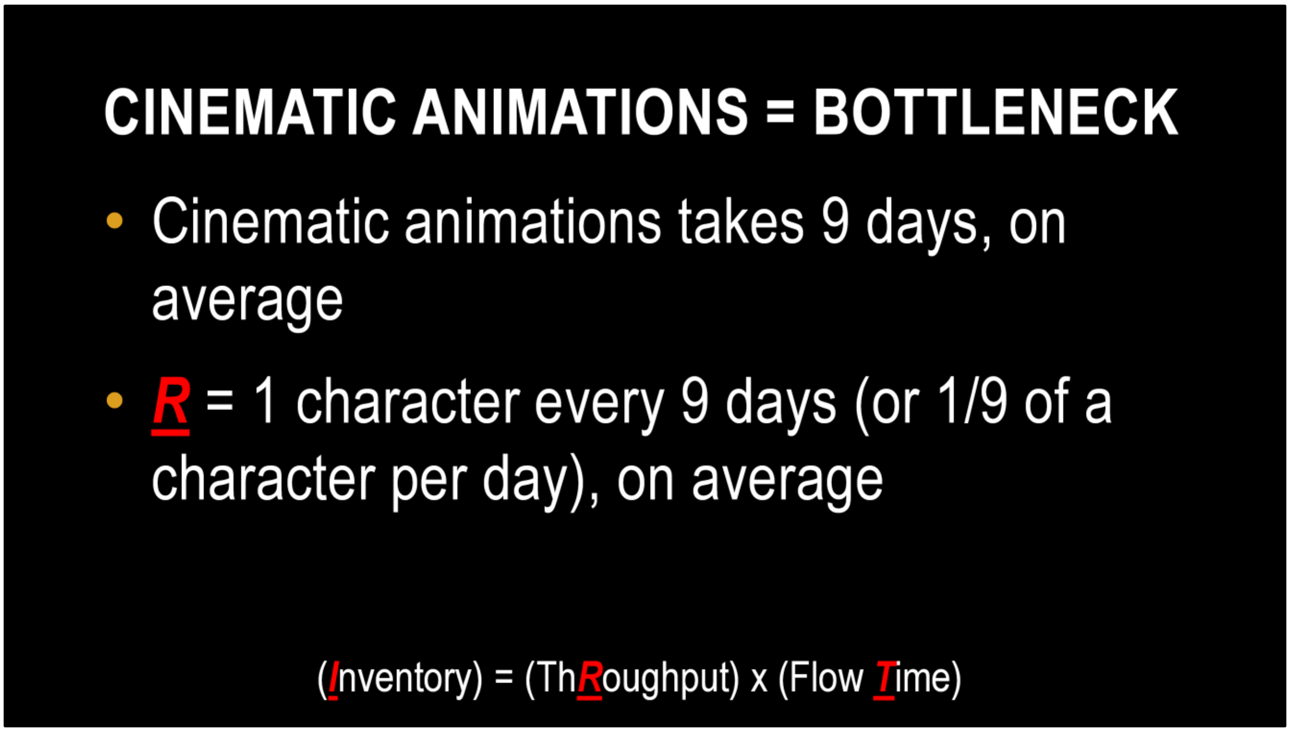
We therefor should expect that an average character takes 29 days to complete.



This slide automatically transitions to the next slide for visual effect



Here is our bottleneck



CINEMATIC ANIMATIONS = BOTTLENECK

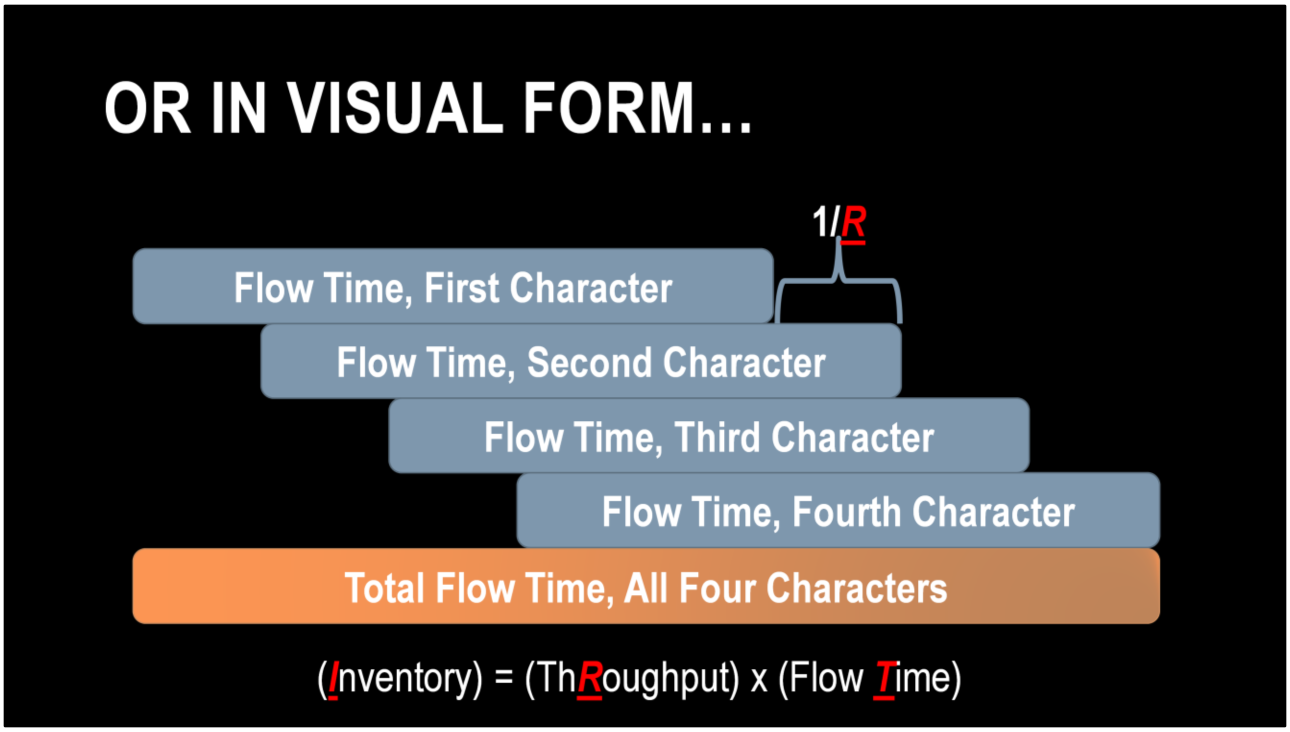
- Cinematic animations takes 9 days, on average
- ***R*** = 1 character every 9 days (or 1/9 of a character per day), on average

(*I*nventory) = (Th*R*oughput) x (Flow *T*ime)

Therefore, we should expect that a complete character will emerge from the process every 9 days on average (or, we can think in terms of 1/9th of a character per day on average).

WITH LITTLE EFFORT, WE ESTIMATED:

- Average time to complete a character from scratch
- Expected time before the first character is in-game
- Average rate of subsequent characters



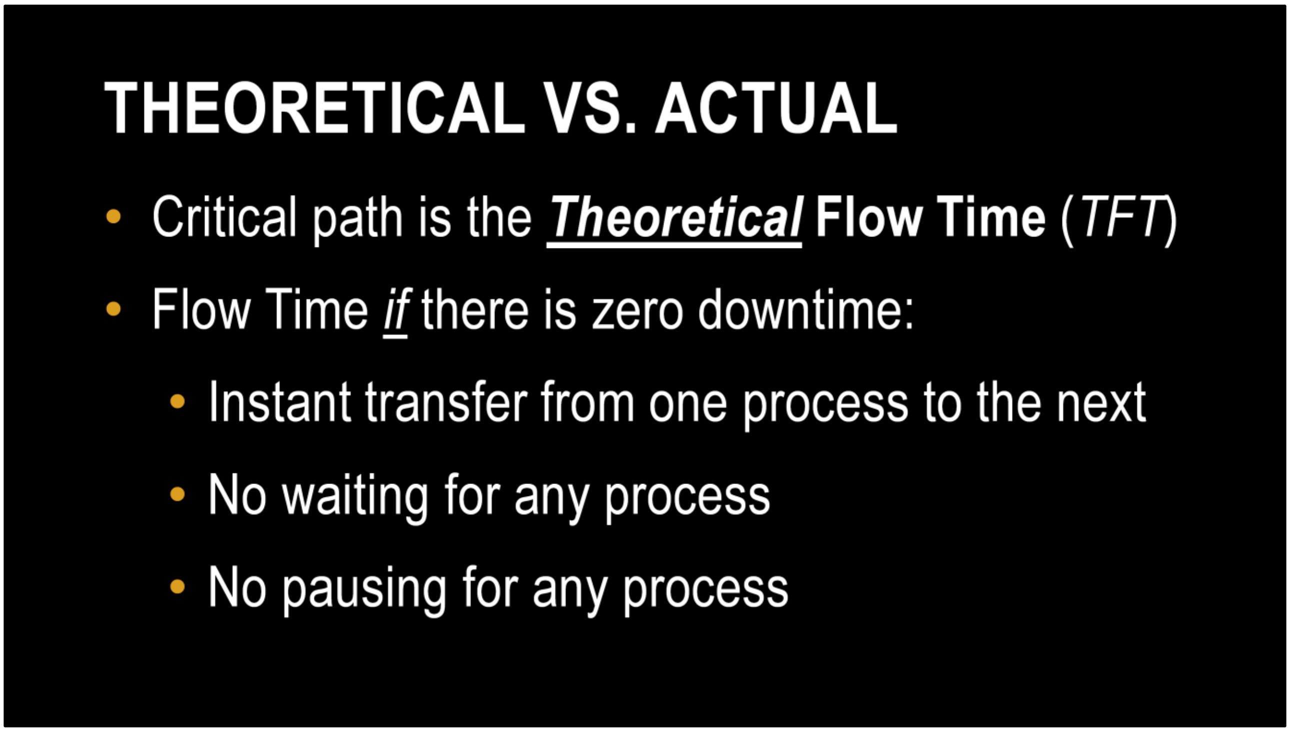
OR IN VISUAL FORM…

1/R
Flow Time, First Character
Flow Time, Second Character
Flow Time, Third Character
Flow Time, Fourth Character
Total Flow Time, All Four Characters

(Inventory) = (ThRoughput) x (Flow Time)

- Blue bars are the time per character
- 1/R is the time between characters
- Orange bar is the total time for all 4 characters

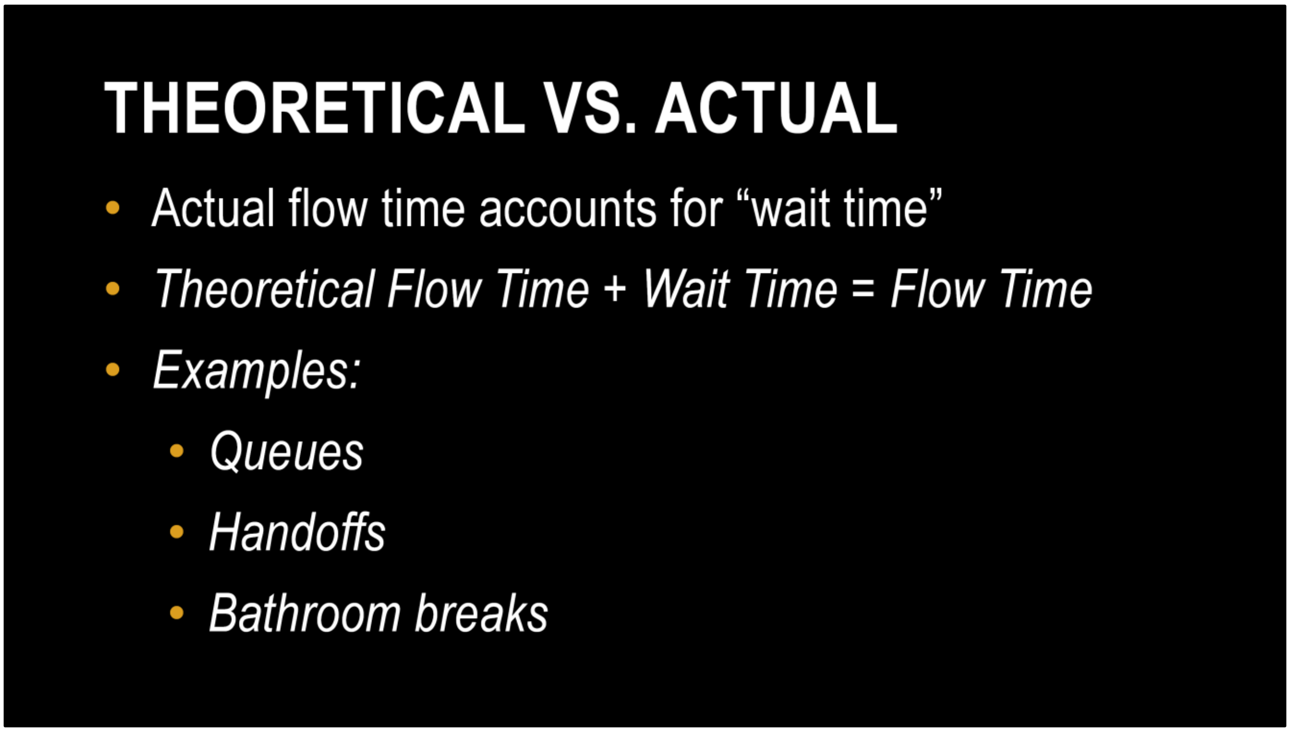
AN IMPORTANT CAVEAT…



THEORETICAL VS. ACTUAL

- Critical path is the ***Theoretical*** **Flow Time** (*TFT*)
- Flow Time *if* there is zero downtime:
  - Instant transfer from one process to the next
  - No waiting for any process
  - No pausing for any process

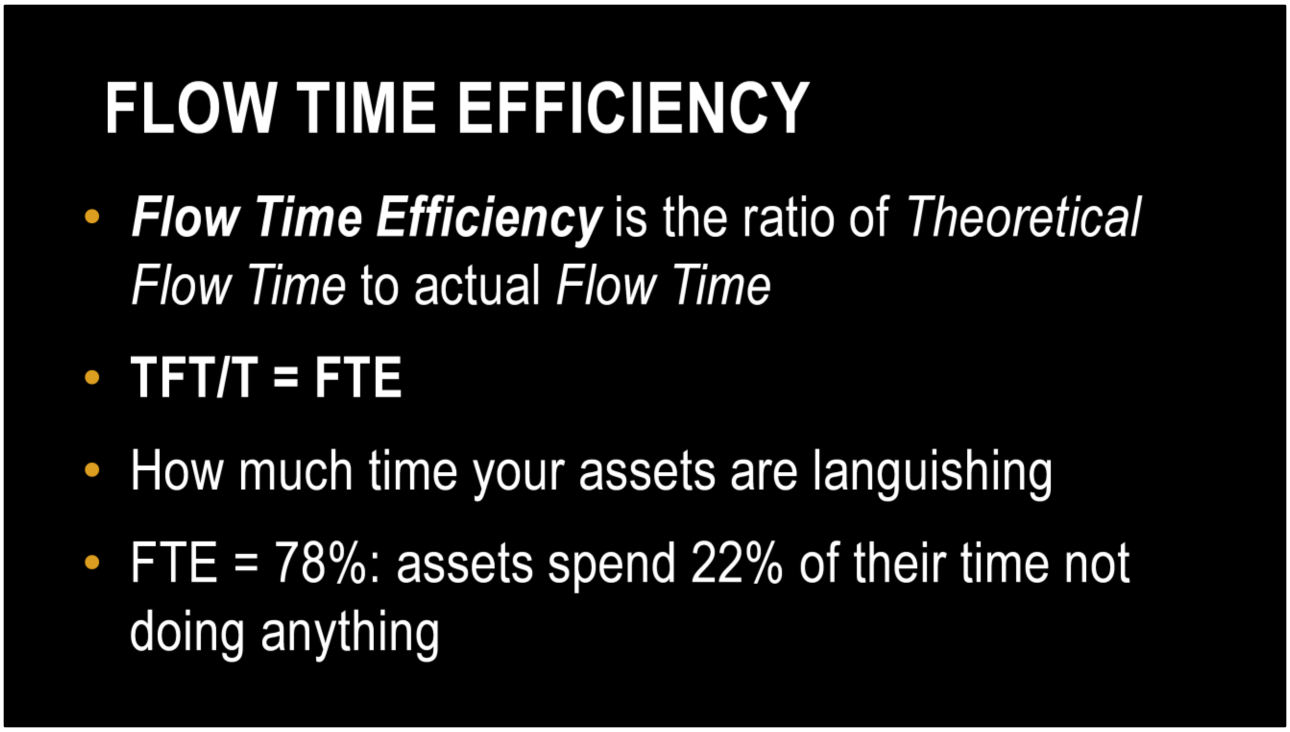
If you determine your flow time by adding the activities in the critical path, you have a theoretical flow time – IE the average flow time if the in-process inventory never has to wait.



THEORETICAL VS. ACTUAL

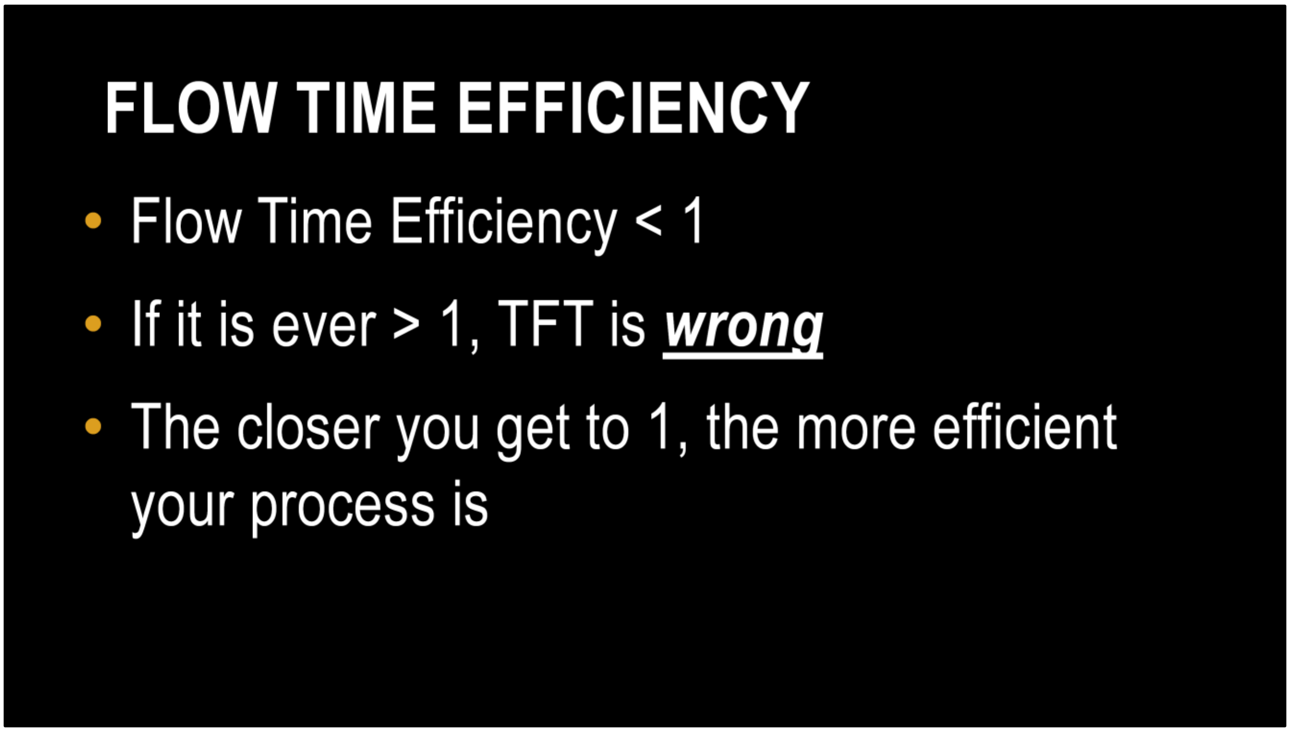
- Actual flow time accounts for “wait time”
- *Theoretical Flow Time + Wait Time = Flow Time*
- *Examples:*
  - *Queues*
  - *Handoffs*
  - *Bathroom breaks*

Anything that causes in-process inventory to stop being processed is Wait Time.



FLOW TIME EFFICIENCY

- ***Flow Time Efficiency*** is the ratio of *Theoretical Flow Time* to actual *Flow Time*
- **TFT/T = FTE**
- How much time your assets are languishing
- FTE = 78%: assets spend 22% of their time not doing anything

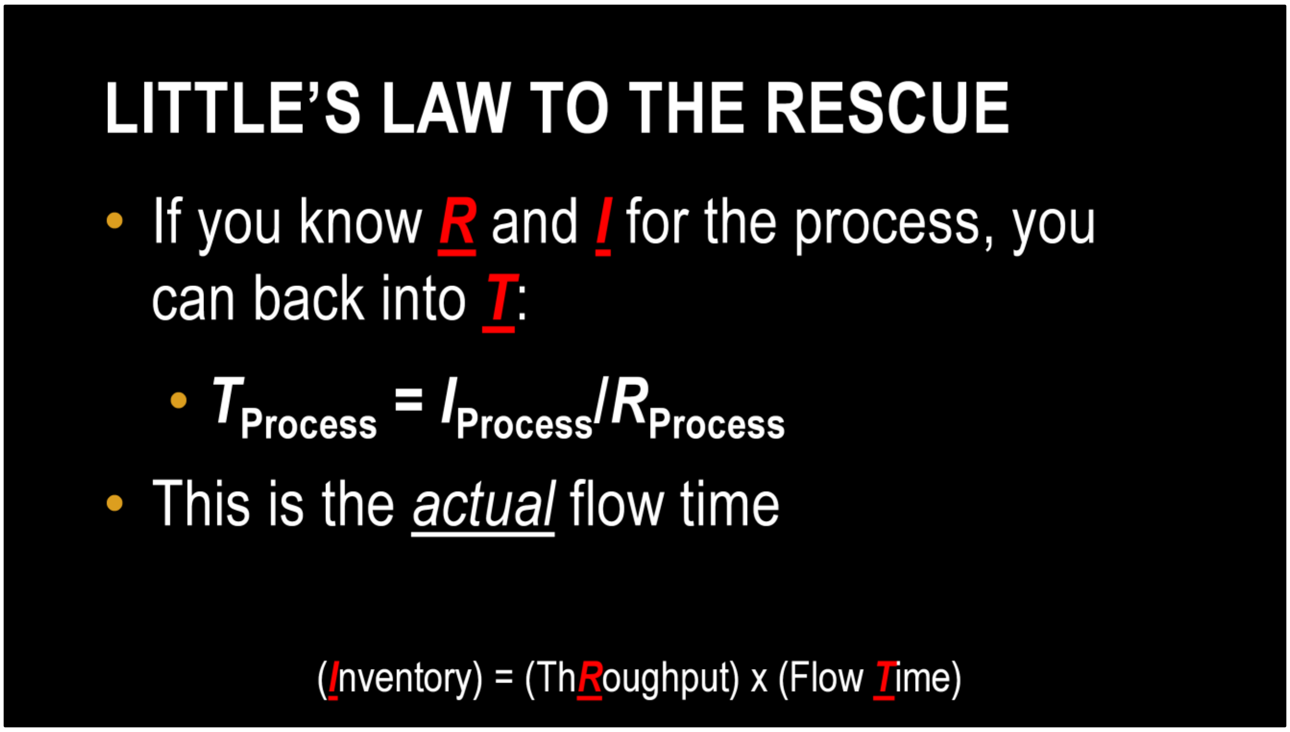


FLOW TIME EFFICIENCY

- Flow Time Efficiency < 1
- If it is ever > 1, TFT is ***wrong***
- The closer you get to 1, the more efficient your process is

If you have an actual flow time that is less that your theoretical flow time, you high-balled the theoretical. IE, you over-estimated how long some or all of your critical activities take. You can’t do better than a process where nothing has to wait.

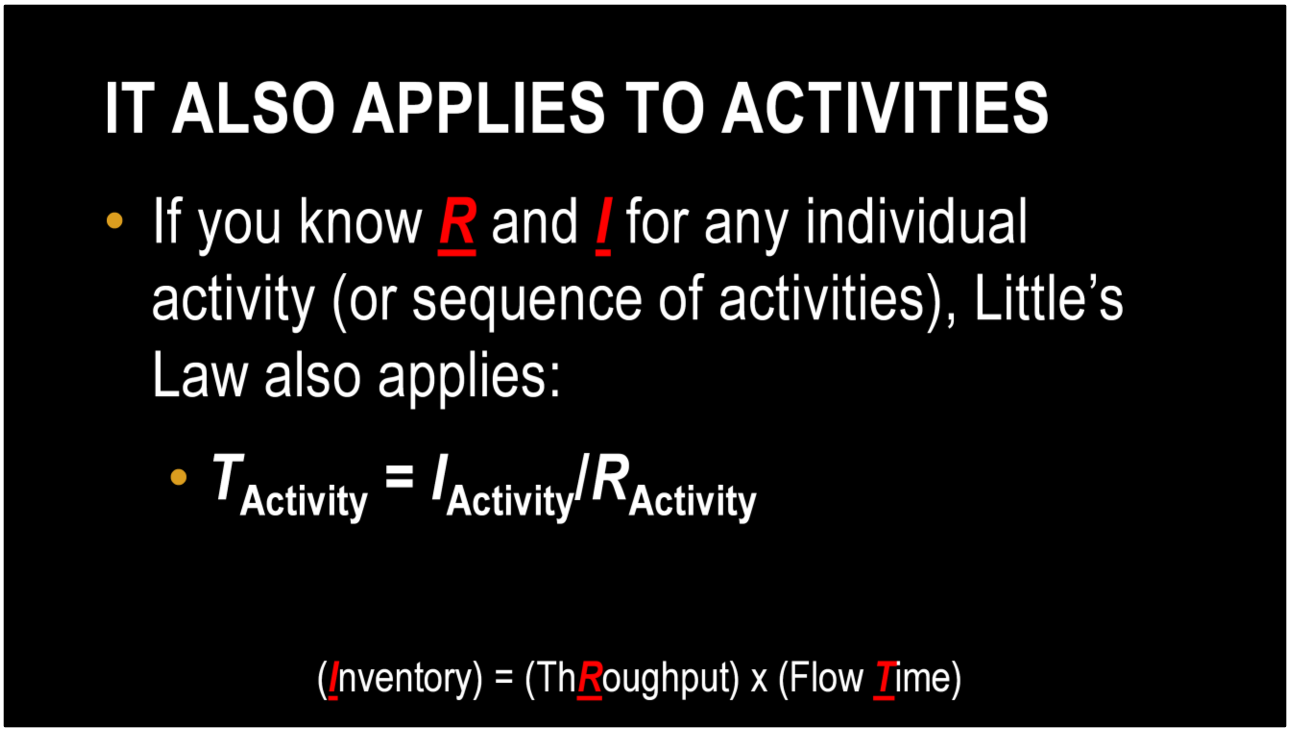
WHAT IF YOU DON’T KNOW THE AVERAGE TIME TO COMPLETE THE ENTIRE PIPELINE, OR ANY SINGLE ACTIVITY?



LITTLE'S LAW TO THE RESCUE

- If you know ***R*** and ***I*** for the process, you can back into ***T***:
  - $T_{\text{Process}} = I_{\text{Process}}/R_{\text{Process}}$
- This is the *actual* flow time

(***I***nventory) = (Th***R***oughput) x (Flow ***T***ime)



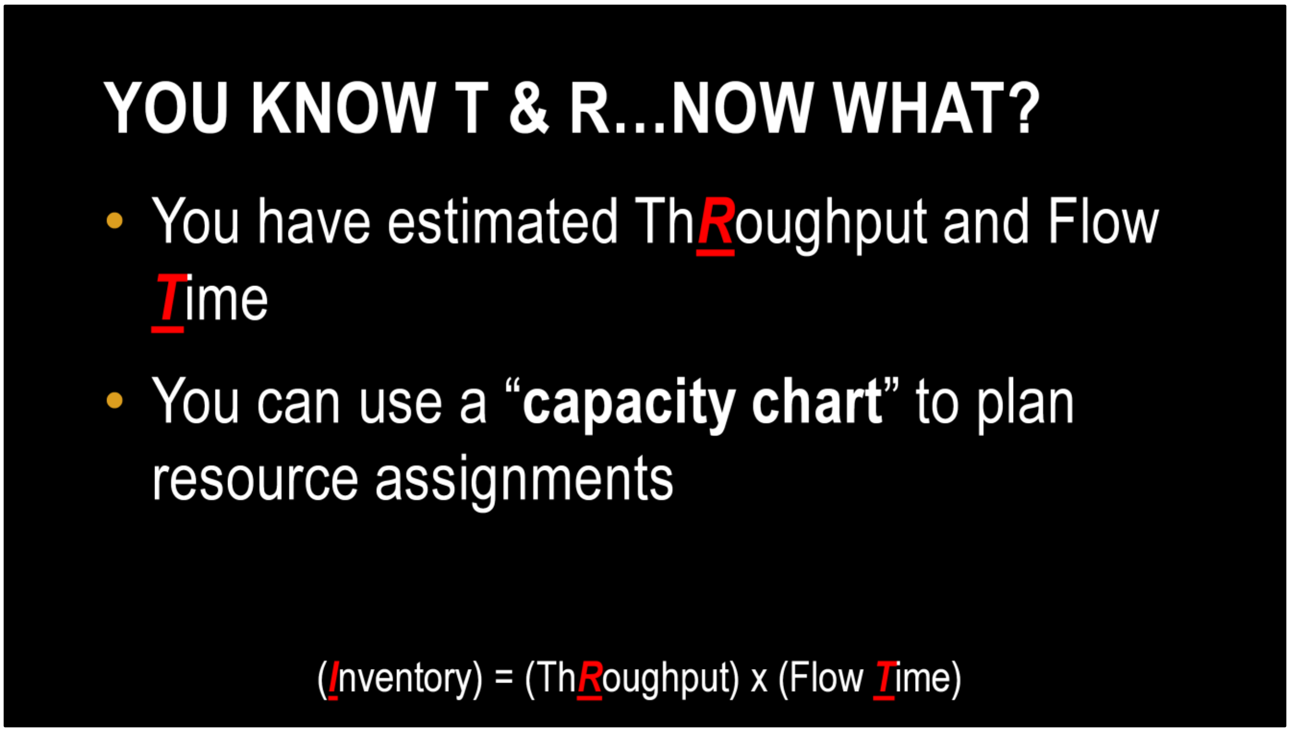
IT ALSO APPLIES TO ACTIVITIES

- If you know ***R*** and ***I*** for any individual activity (or sequence of activities), Little's Law also applies:
  - $T_{\text{Activity}} = I_{\text{Activity}}/R_{\text{Activity}}$

(*I*nventory) = (Th*R*oughput) x (Flow *T*ime)

To, emphasize the flexibility of Little's Law: it can apply to any level of granularity - the whole pipeline, continuous sections of the pipeline, or individual processes

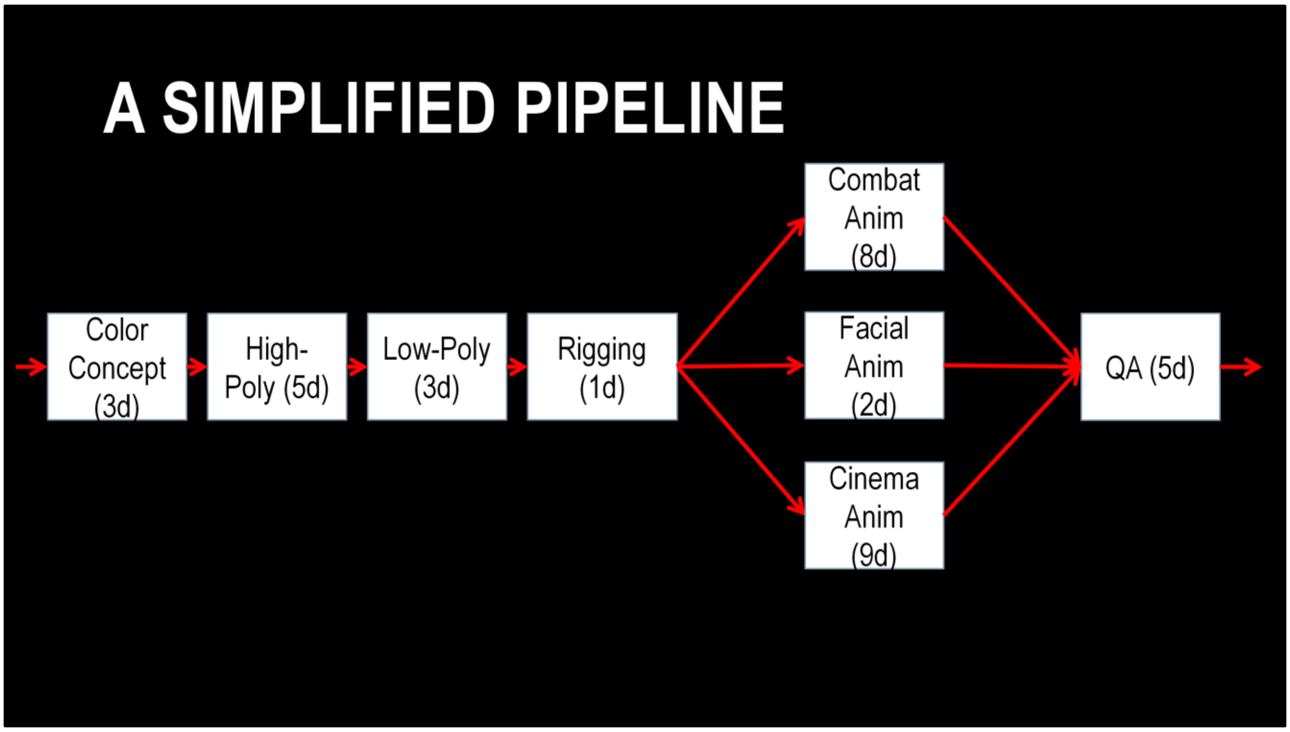
OPTIMALLY STAFFING PIPELINES USING CAPACITY CHARTS



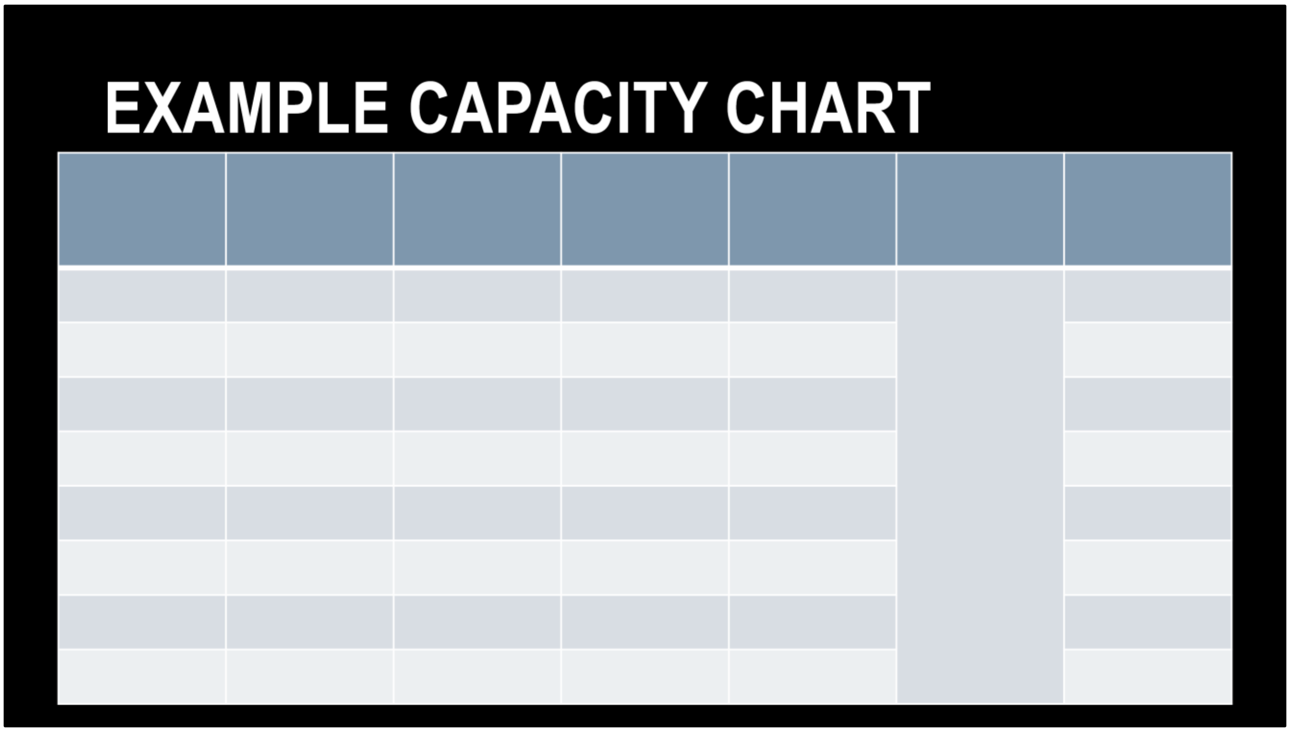
YOU KNOW T & R…NOW WHAT?

- You have estimated Th***R***oughput and Flow ***T***ime
- You can use a "**capacity chart**" to plan resource assignments

(***I***nventory) = (Th***R***oughput) x (Flow ***T***ime)

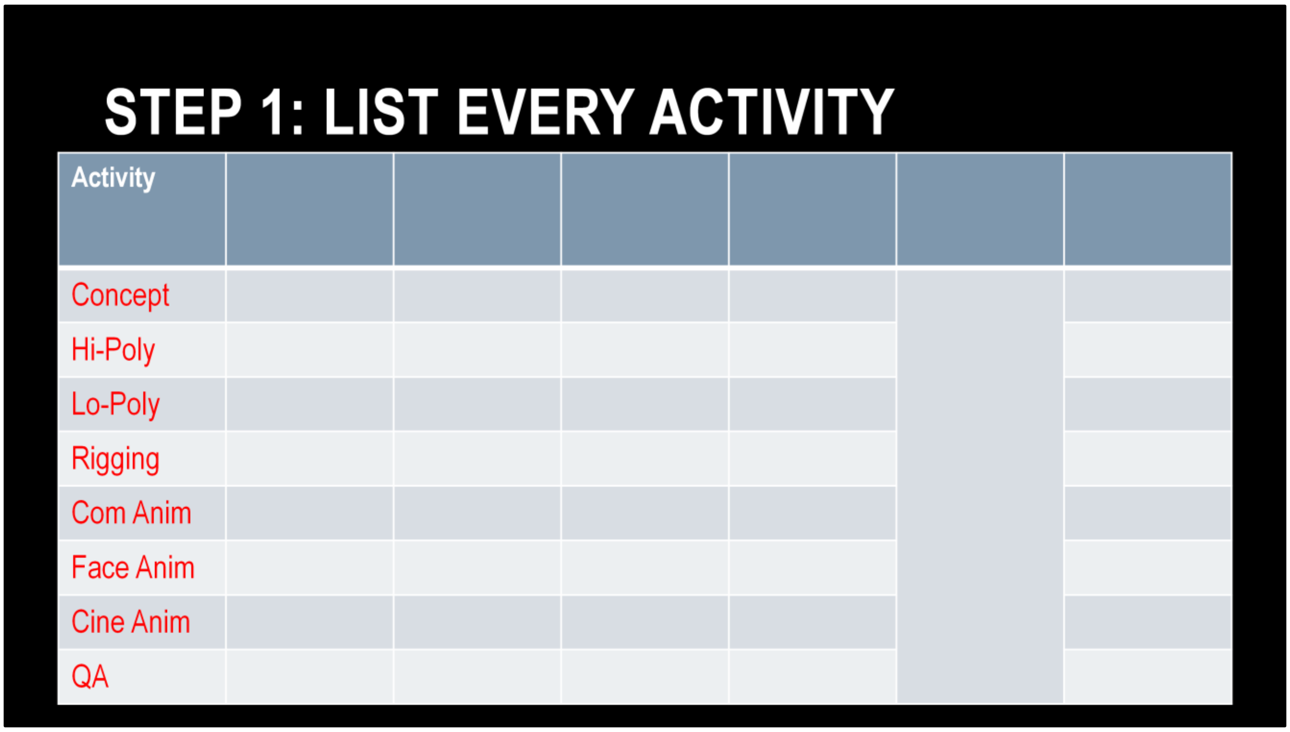

A SIMPLIFIED PIPELINE

I'm simplifying the previous pipeline for the sake of clarity. But a capacity chart would still work on the more complicated pipeline from the last example.



EXAMPLE CAPACITY CHART

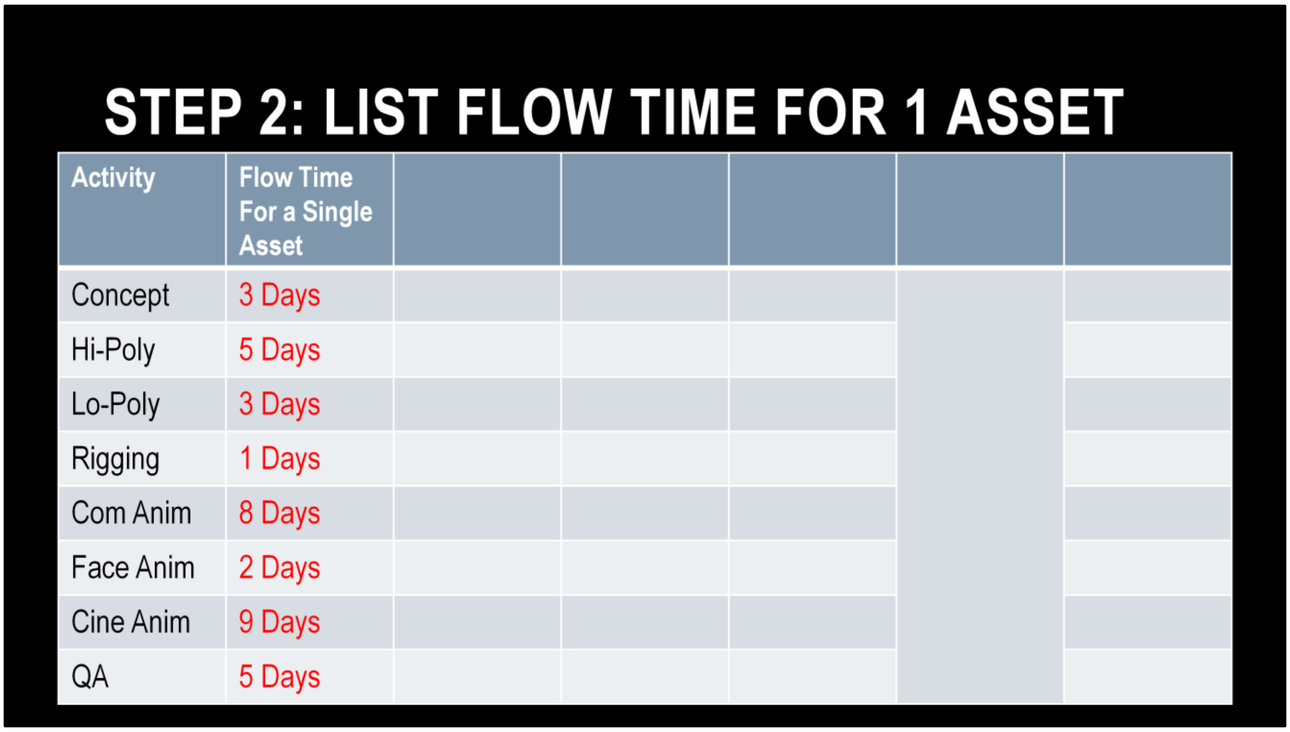
Go into Excel and create a simple 7-column chart (or just download a copy from the link at the end of the slides, before the Appendix)



STEP 1: LIST EVERY ACTIVITY

Activity						
Concept						
Hi-Poly						
Lo-Poly						
Rigging						
Com Anim						
Face Anim						
Cine Anim						
QA						

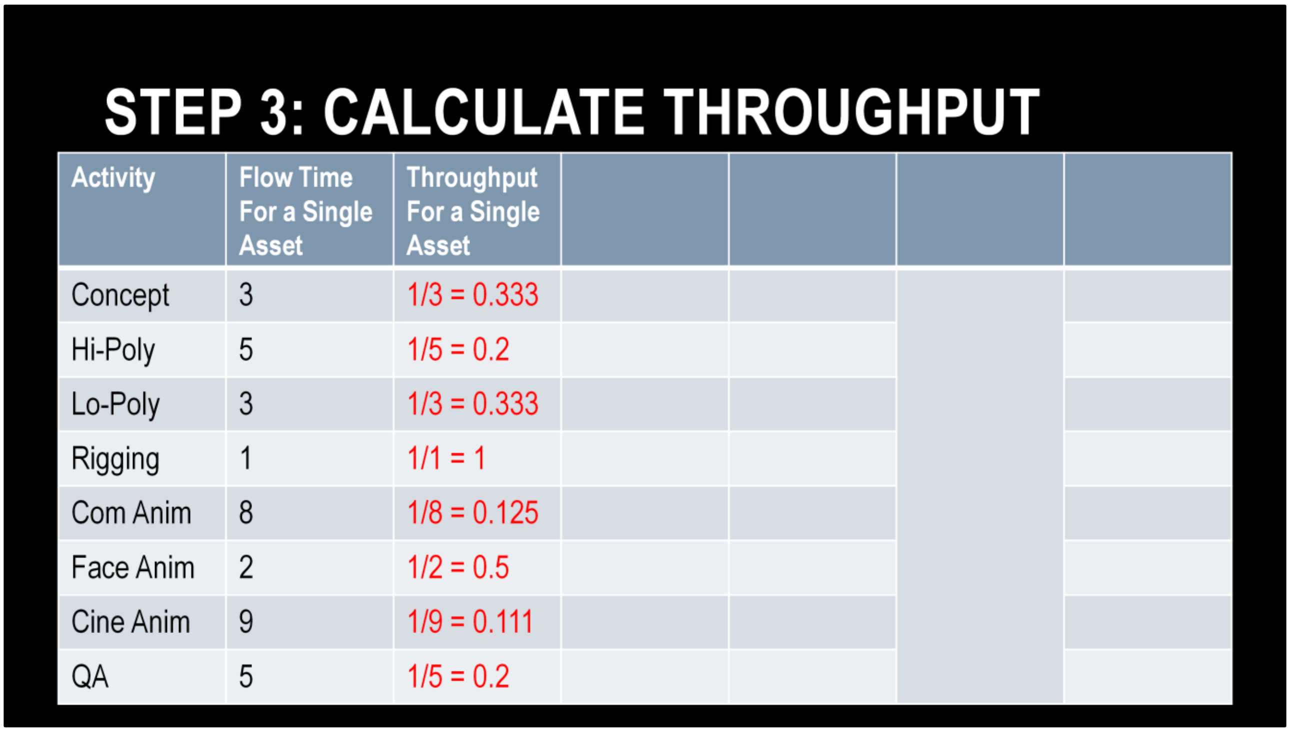
First, list every activity in the pipeline, critical path or otherwise. Order doesn't matter.



STEP 2: LIST FLOW TIME FOR 1 ASSET

Activity	Flow Time For a Single Asset					
Concept	3 Days					
Hi-Poly	5 Days					
Lo-Poly	3 Days					
Rigging	1 Days					
Com Anim	8 Days					
Face Anim	2 Days					
Cine Anim	9 Days					
QA	5 Days					

Next, list the average time it takes to complete a pass of each activity for a single asset

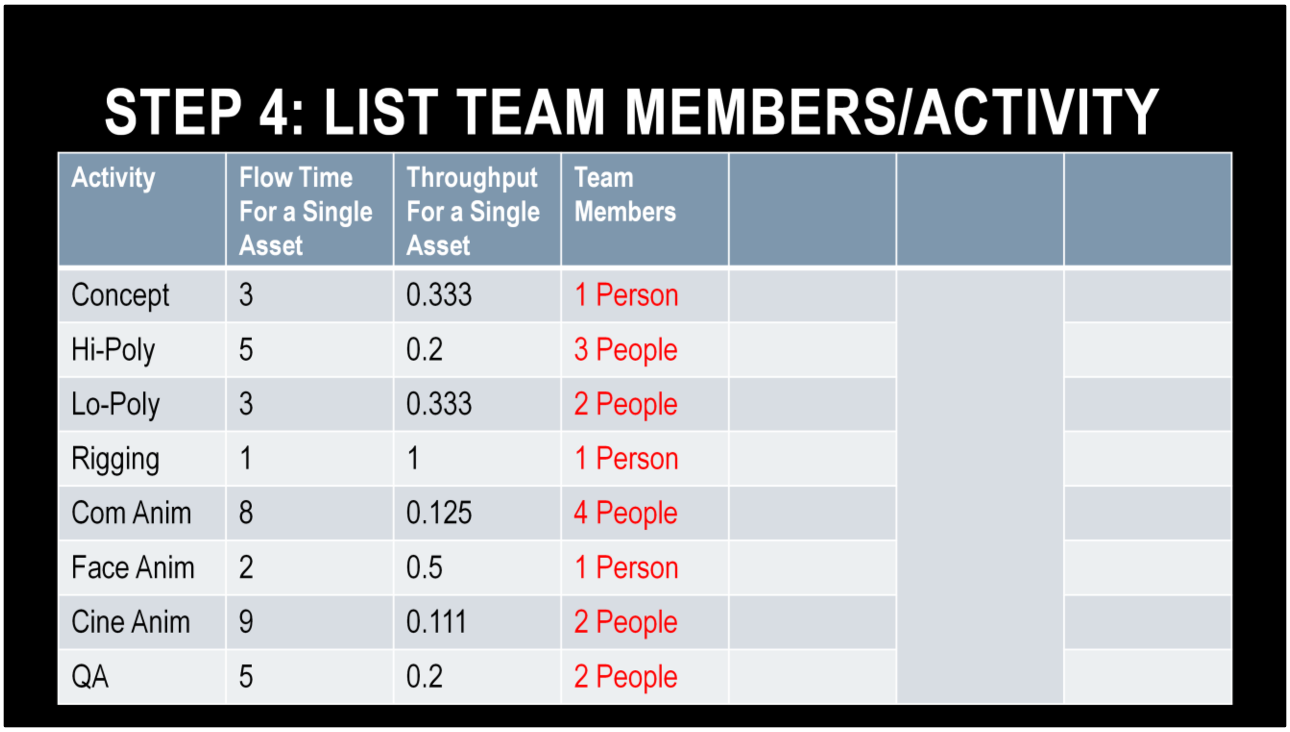

STEP 3: CALCULATE THROUGHPUT

Activity	Flow Time For a Single Asset	Throughput For a Single Asset				
Concept	3	1/3 = 0.333				
Hi-Poly	5	1/5 = 0.2				
Lo-Poly	3	1/3 = 0.333				
Rigging	1	1/1 = 1				
Com Anim	8	1/8 = 0.125				
Face Anim	2	1/2 = 0.5				
Cine Anim	9	1/9 = 0.111				
QA	5	1/5 = 0.2				

Next calculate the single asset throughput (the rate at which single units emerge from the activity) by taking the inverse of the activity time.

To explain how this works, if you take Little's Law and assume Inventory = 1 (because we are talking about the flow time for a single unit), then Flow Time and Throughput are reciprocal:

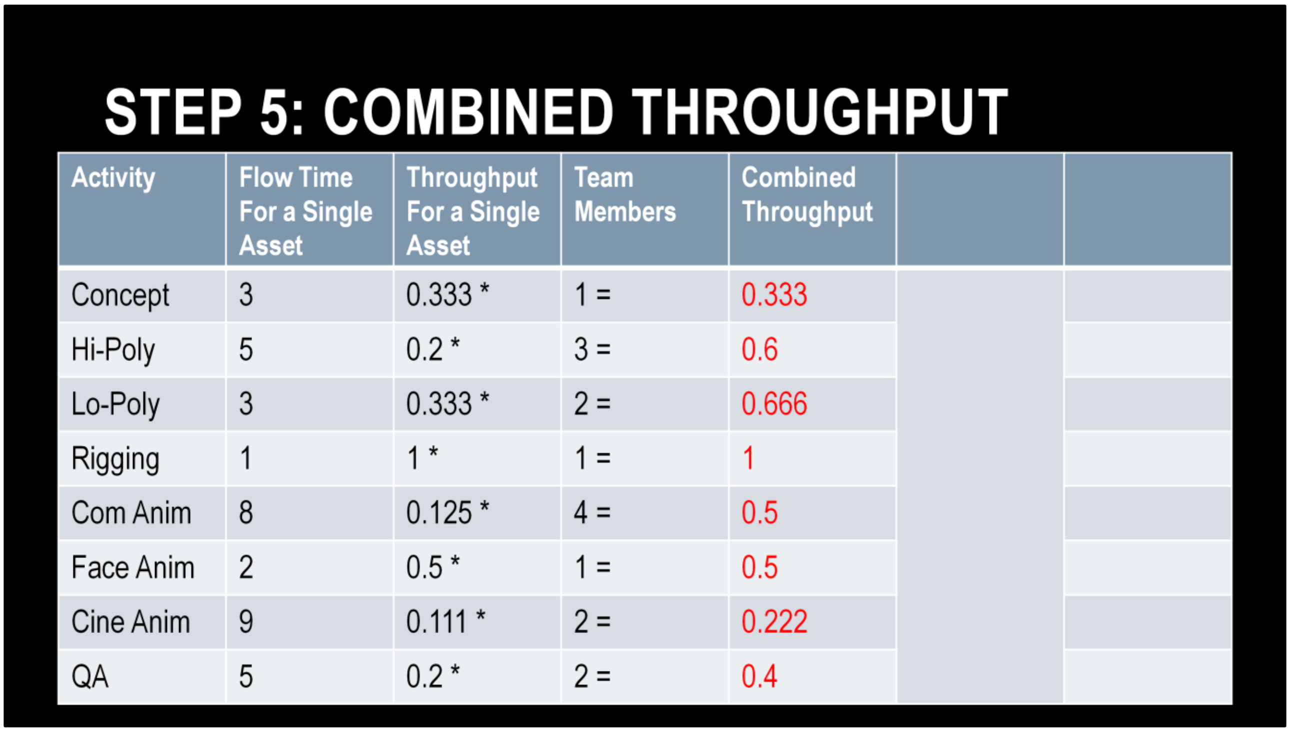
Inventory = Throughput * Flow Time
Flow Time = Inventory/Throughput
Inventory = 1
Flow Time = 1/Throughput



STEP 4: LIST TEAM MEMBERS/ACTIVITY

Activity	Flow Time For a Single Asset	Throughput For a Single Asset	Team Members			
Concept	3	0.333	1 Person			
Hi-Poly	5	0.2	3 People			
Lo-Poly	3	0.333	2 People			
Rigging	1	1	1 Person			
Com Anim	8	0.125	4 People			
Face Anim	2	0.5	1 Person			
Cine Anim	9	0.111	2 People			
QA	5	0.2	2 People			

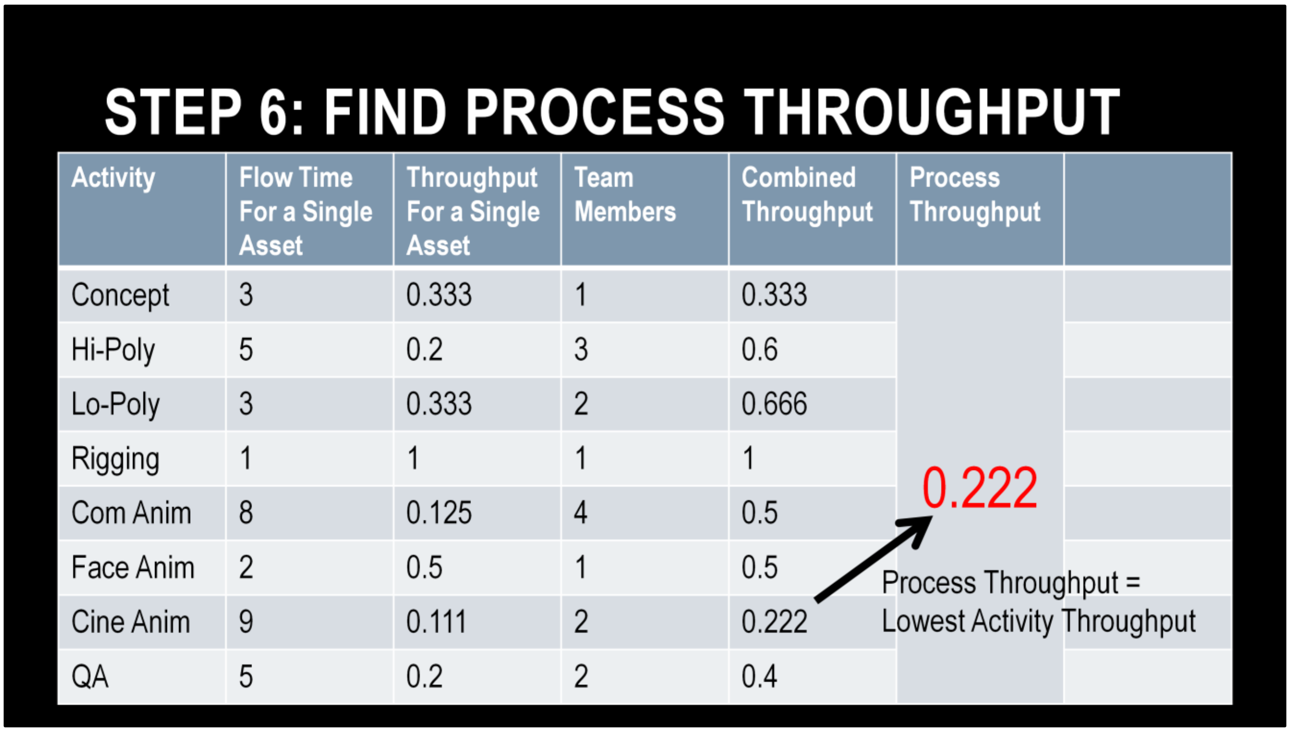
Now, list the number of people you have performing each activity



STEP 5: COMBINED THROUGHPUT

Activity	Flow Time For a Single Asset	Throughput For a Single Asset	Team Members	Combined Throughput		
Concept	3	0.333 *	1 =	0.333		
Hi-Poly	5	0.2 *	3 =	0.6		
Lo-Poly	3	0.333 *	2 =	0.666		
Rigging	1	1 *	1 =	1		
Com Anim	8	0.125 *	4 =	0.5		
Face Anim	2	0.5 *	1 =	0.5		
Cine Anim	9	0.111 *	2 =	0.222		
QA	5	0.2 *	2 =	0.4		

Multiply the throughput per activity by the number of people performing the activity to calculate the combined throughput for that activity. This is the collective average throughput of all of the people handling the activity.

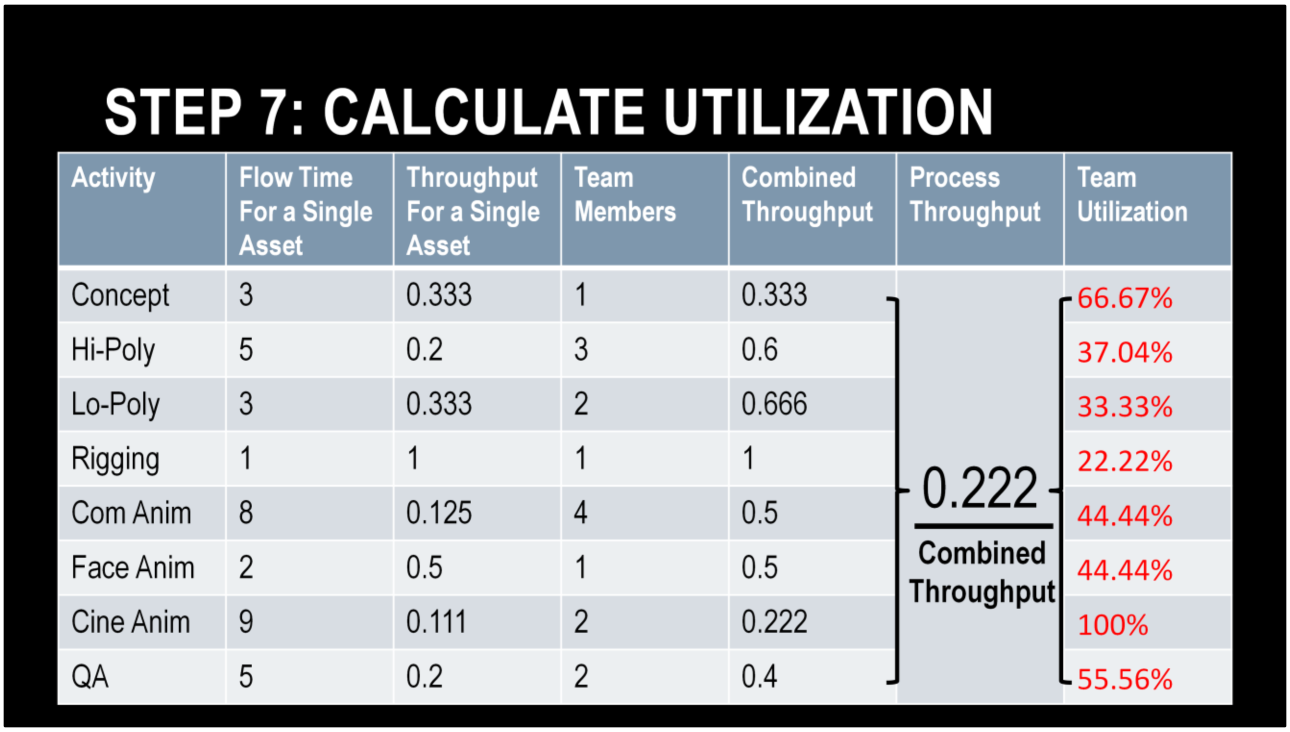


STEP 6: FIND PROCESS THROUGHPUT

Activity	Flow Time For a Single Asset	Throughput For a Single Asset	Team Members	Combined Throughput	Process Throughput	
Concept	3	0.333	1	0.333		
Hi-Poly	5	0.2	3	0.6		
Lo-Poly	3	0.333	2	0.666		
Rigging	1	1	1	1		
Com Anim	8	0.125	4	0.5	0.222	
Face Anim	2	0.5	1	0.5		
Cine Anim	9	0.111	2	0.222		
QA	5	0.2	2	0.4		

Process Throughput = Lowest Activity Throughput

Identify the activity that has the lowest combined throughput. As established in the Alien Queen example, the activity (or activities) with the lowest throughput – the bottleneck(s) – dictate the throughput for the entire process.

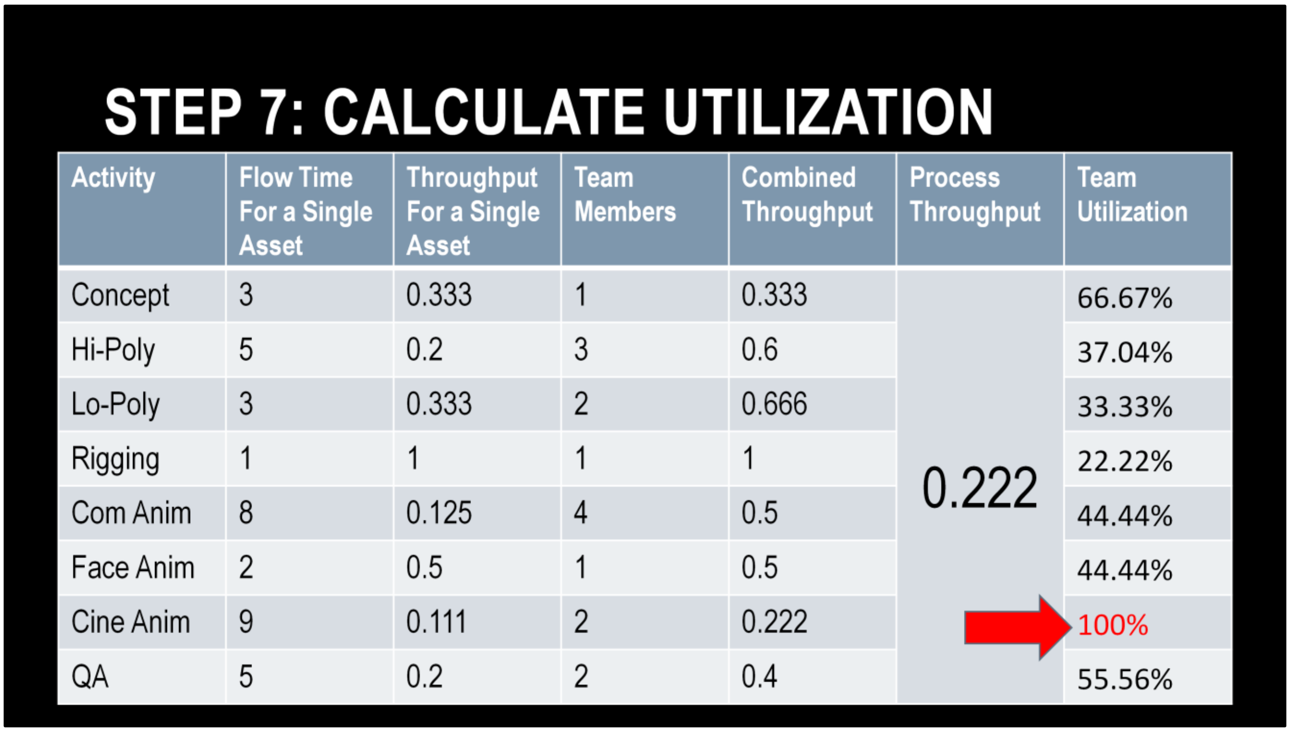

STEP 7: CALCULATE UTILIZATION

Activity	Flow Time For a Single Asset	Throughput For a Single Asset	Team Members	Combined Throughput	Process Throughput	Team Utilization
Concept	3	0.333	1	0.333	0.222 Combined Throughput	66.67%
Hi-Poly	5	0.2	3	0.6		37.04%
Lo-Poly	3	0.333	2	0.666		33.33%
Rigging	1	1	1	1		22.22%
Com Anim	8	0.125	4	0.5		44.44%
Face Anim	2	0.5	1	0.5		44.44%
Cine Anim	9	0.111	2	0.222		100%
QA	5	0.2	2	0.4		55.56%

Finally, divide the process capacity by each individual combined throughput to calculate the respective utilization per activity. This number tells you how much of a given team's bandwidth this process will consume.

- Examples:
    - This process will consume 2/3's of our concept artist's bandwidth
    - And 1/3 of our low-poly modelers' collective bandwidth
    - And 100% of the cinematic animators' collective bandwidth

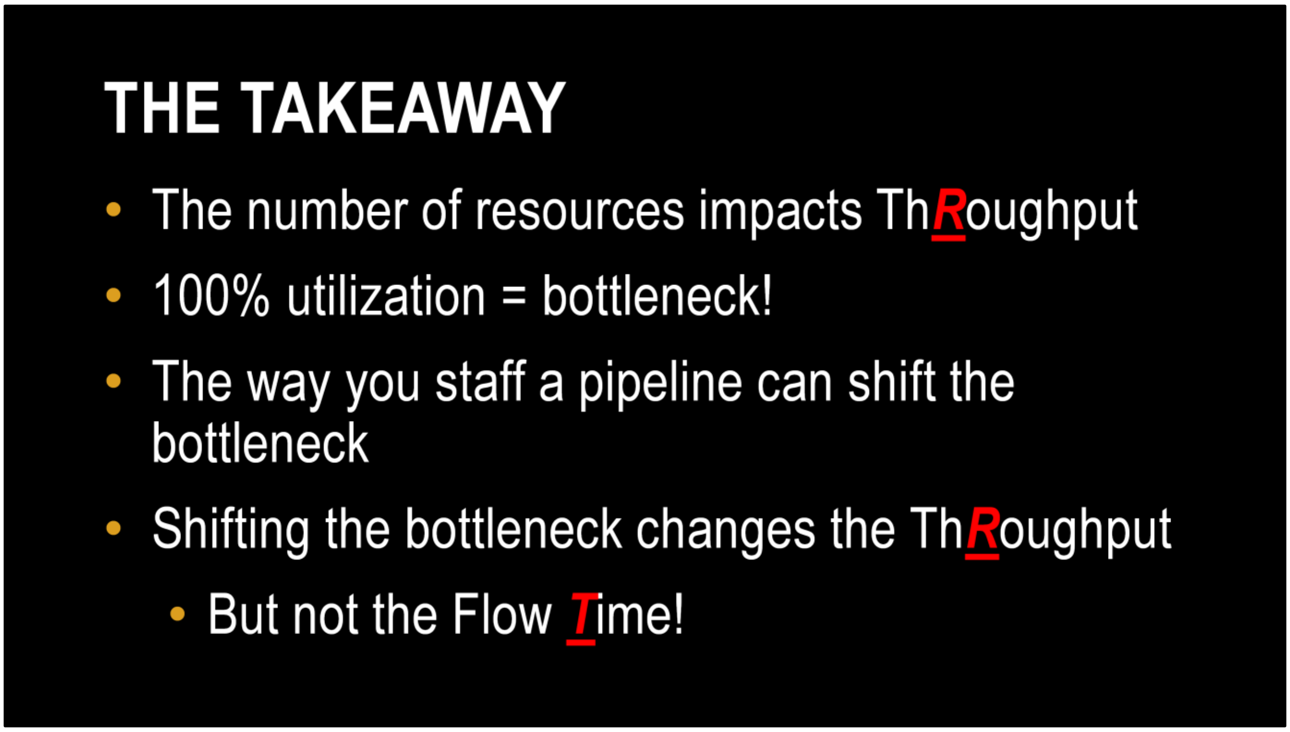
It's important to note that the higher the percentage, the closer to pool is to being a bottleneck.



STEP 7: CALCULATE UTILIZATION

Activity	Flow Time For a Single Asset	Throughput For a Single Asset	Team Members	Combined Throughput	Process Throughput	Team Utilization
Concept	3	0.333	1	0.333	0.222	66.67%
Hi-Poly	5	0.2	3	0.6		37.04%
Lo-Poly	3	0.333	2	0.666		33.33%
Rigging	1	1	1	1		22.22%
Com Anim	8	0.125	4	0.5		44.44%
Face Anim	2	0.5	1	0.5		44.44%
Cine Anim	9	0.111	2	0.222		100%
QA	5	0.2	2	0.4		55.56%

Any pool with 100% utilization is a bottleneck.

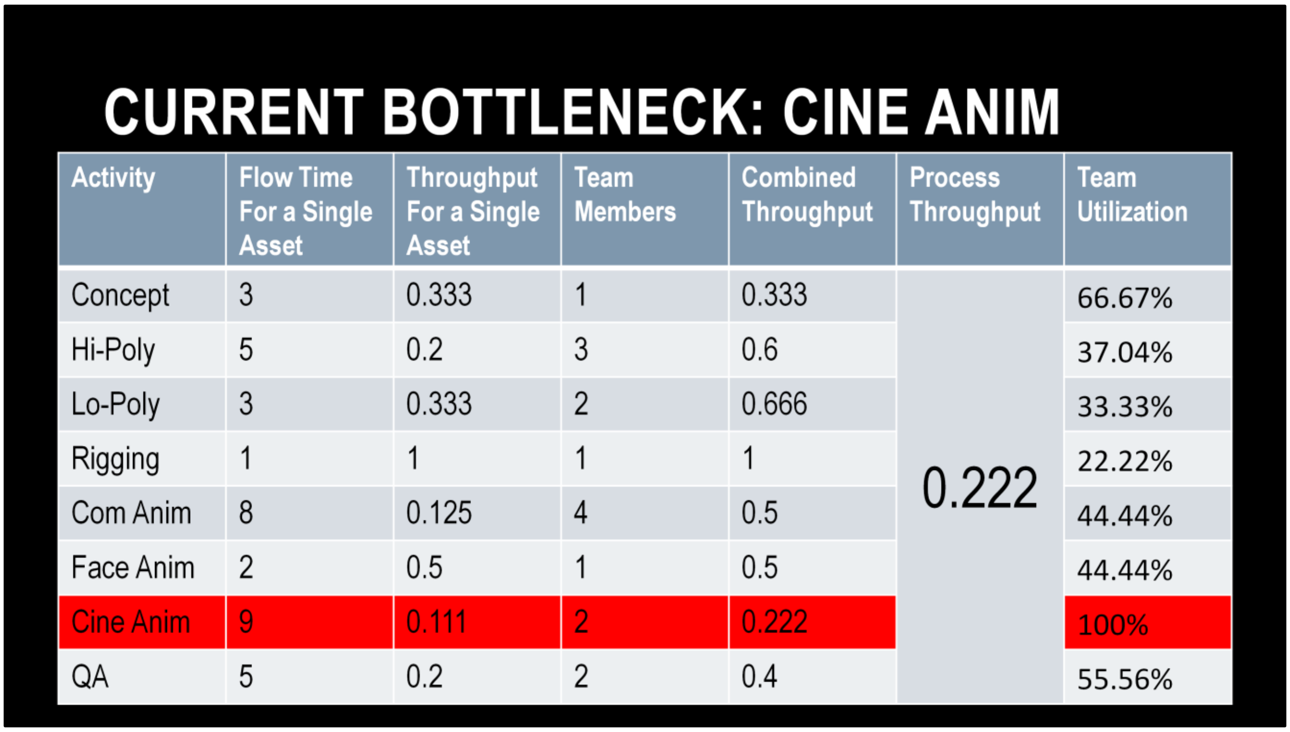


THE TAKEAWAY

- The number of resources impacts Th***R***oughput
- 100% utilization = bottleneck!
- The way you staff a pipeline can shift the bottleneck
- Shifting the bottleneck changes the Th***R***oughput
  - But not the Flow ***T***ime!

On that last bullet: don’t fall for the “one month baby” trap – you can’t throw a ton of people at the same asset and get it faster. You can divvy up a list of animations amongst animators to some extent, but you cant throw three character modelers at the same model and get it done in a third of the time.

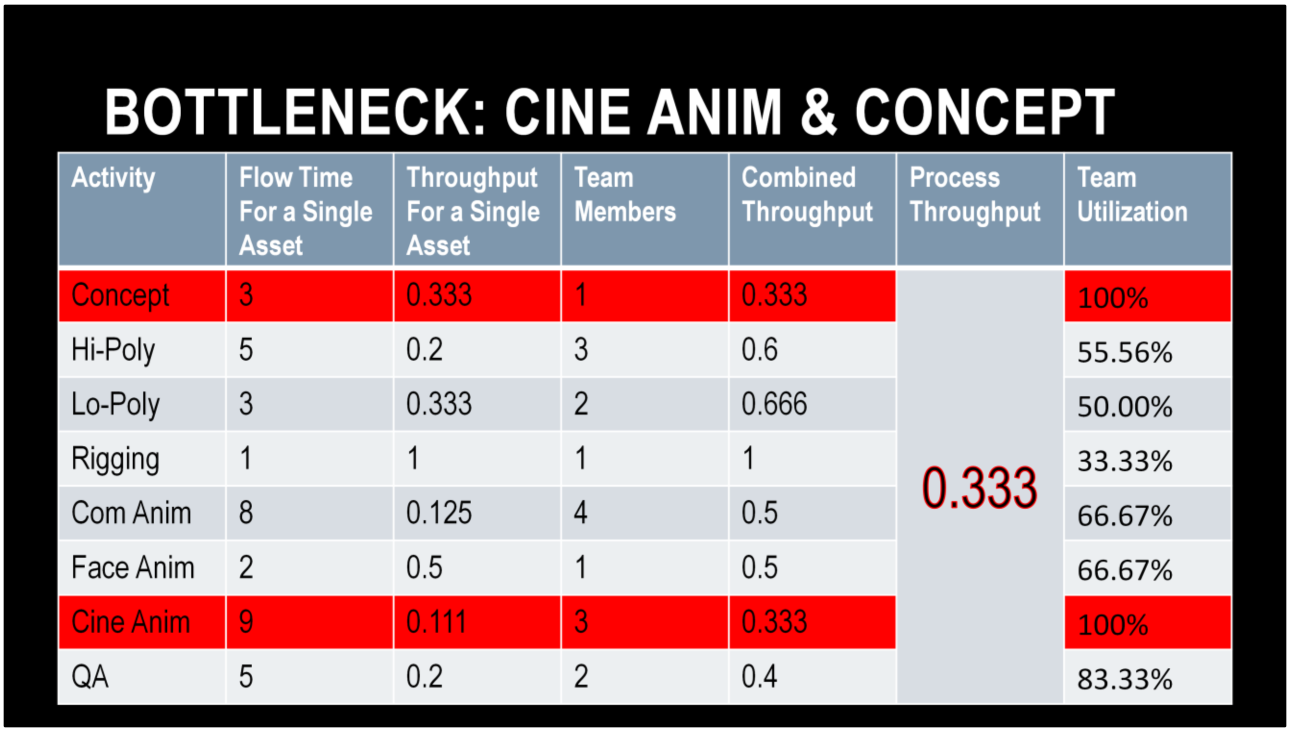
WHAT HAPPENS IF WE ADD A RESOURCE TO THE BOTTLENECK?



CURRENT BOTTLENECK: CINE ANIM

Activity	Flow Time For a Single Asset	Throughput For a Single Asset	Team Members	Combined Throughput	Process Throughput	Team Utilization
Concept	3	0.333	1	0.333	0.222	66.67%
Hi-Poly	5	0.2	3	0.6		37.04%
Lo-Poly	3	0.333	2	0.666		33.33%
Rigging	1	1	1	1		22.22%
Com Anim	8	0.125	4	0.5		44.44%
Face Anim	2	0.5	1	0.5		44.44%
Cine Anim	9	0.111	2	0.222		100%
QA	5	0.2	2	0.4		55.56%

Again, here is the same capacity chart, with cinematic animations as the bottleneck. Let’s add one team member to the cinematic animation pool.

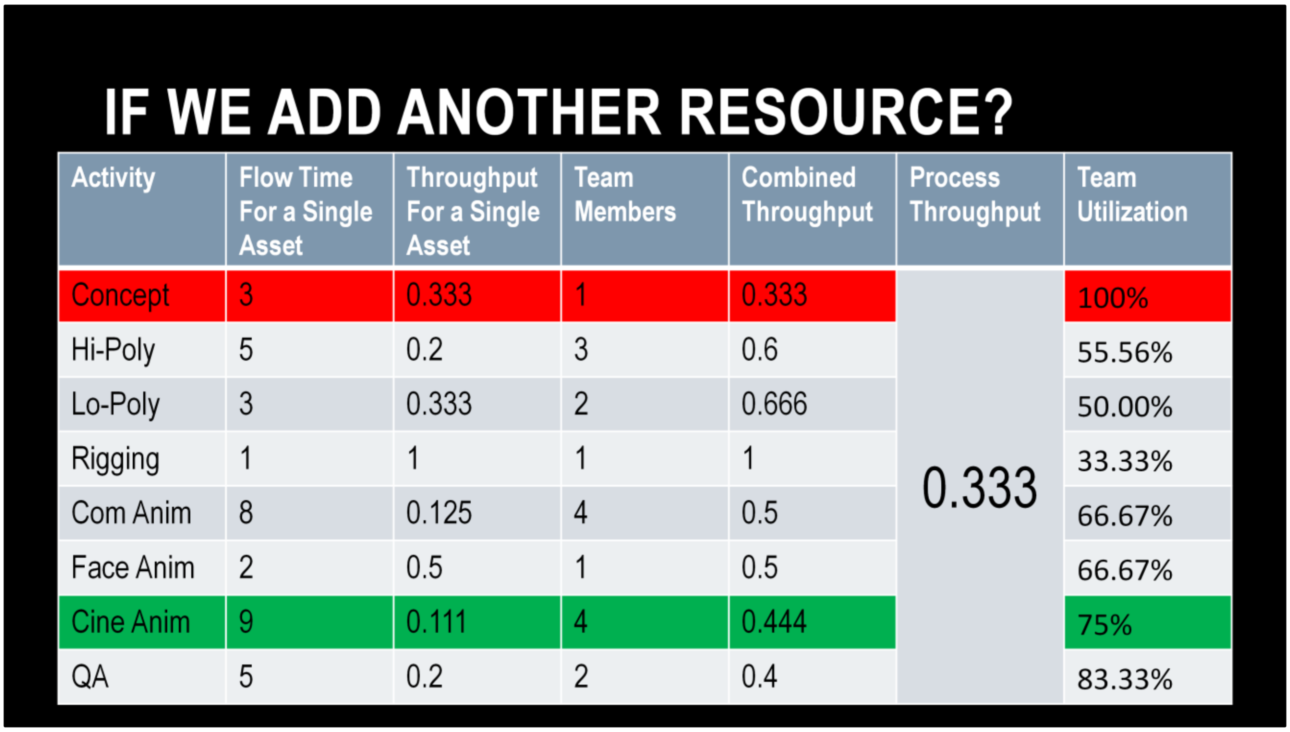


BOTTLENECK: CINE ANIM & CONCEPT

Activity	Flow Time For a Single Asset	Throughput For a Single Asset	Team Members	Combined Throughput	Process Throughput	Team Utilization
Concept	3	0.333	1	0.333	0.333	100%
Hi-Poly	5	0.2	3	0.6		55.56%
Lo-Poly	3	0.333	2	0.666		50.00%
Rigging	1	1	1	1		33.33%
Com Anim	8	0.125	4	0.5		66.67%
Face Anim	2	0.5	1	0.5		66.67%
Cine Anim	9	0.111	3	0.333		100%
QA	5	0.2	2	0.4		83.33%

Two things happened:

1) While cinematic animation is still the bottleneck, throughput has increased to 1/3 of a character per day. In turn, overall throughput has also increased to 1/3 of a character per day
2) Concept art is now also a bottleneck, so we now have two bottlenecks. We aren't worse off because we have two bottlenecks – we're still moving faster than before. But it means we cannot improve throughput by adding a single person.

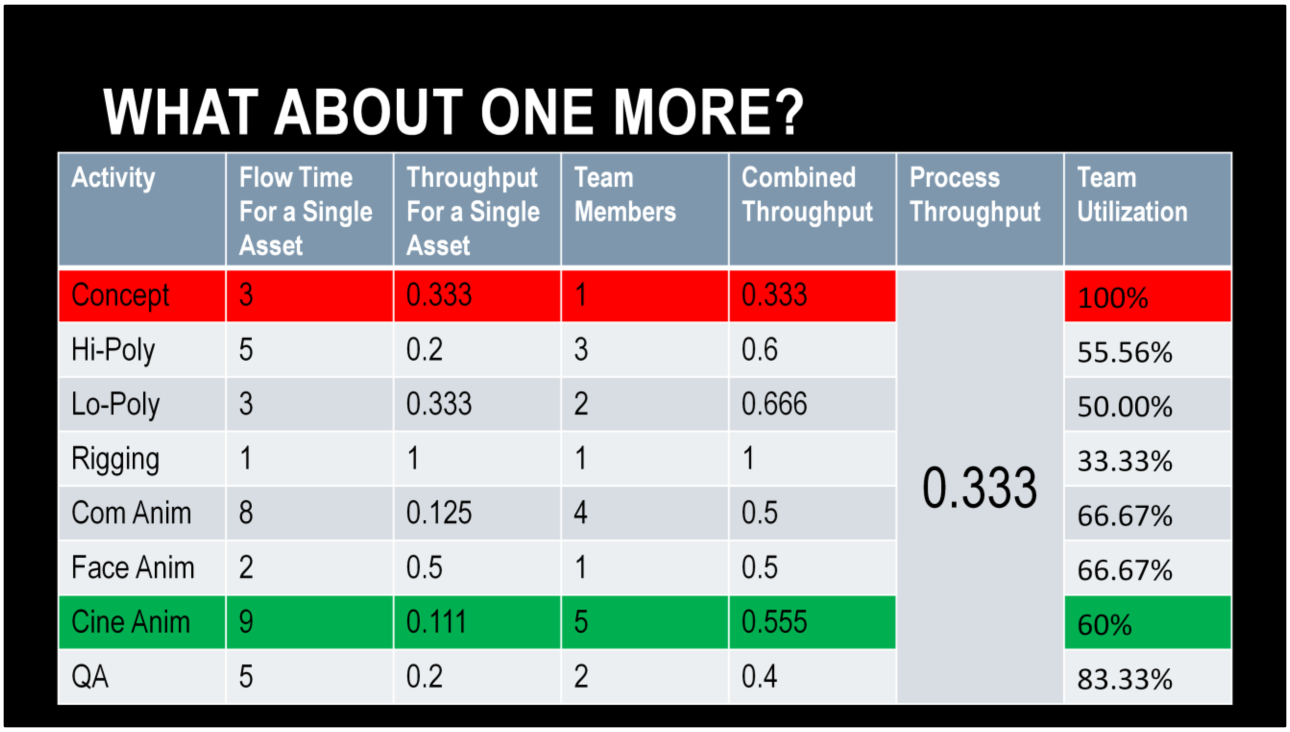


IF WE ADD ANOTHER RESOURCE?

Activity	Flow Time For a Single Asset	Throughput For a Single Asset	Team Members	Combined Throughput	Process Throughput	Team Utilization
Concept	3	0.333	1	0.333	0.333	100%
Hi-Poly	5	0.2	3	0.6		55.56%
Lo-Poly	3	0.333	2	0.666		50.00%
Rigging	1	1	1	1		33.33%
Com Anim	8	0.125	4	0.5		66.67%
Face Anim	2	0.5	1	0.5		66.67%
Cine Anim	9	0.111	4	0.444		75%
QA	5	0.2	2	0.4		83.33%

For example, if we add another cinematic animator, two more things happen:

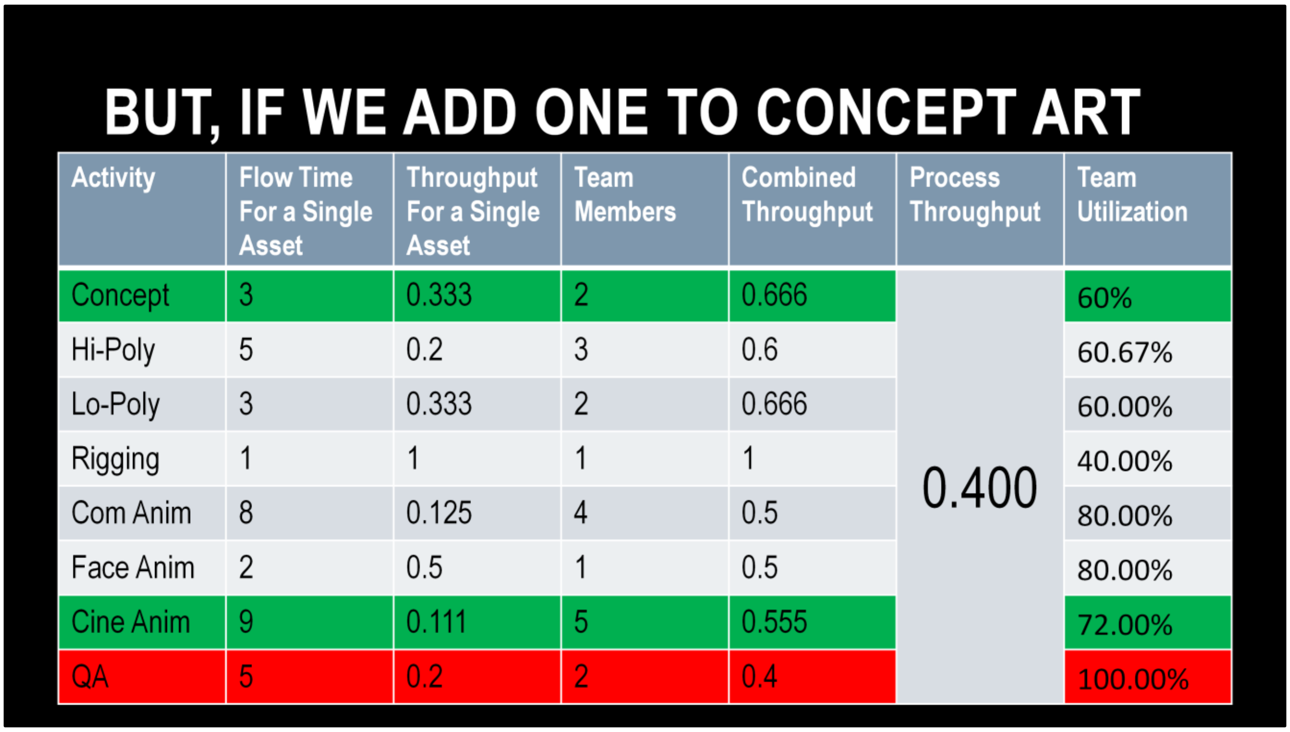
1) Cinematic animation is no longer a bottleneck, and now has some spare capacity (utilization < 100%)
2) But Concept art still is, so overall throughput is still capped at 1/3 character/day; IE, that new resource did not improve throughput



WHAT ABOUT ONE MORE?

Activity	Flow Time For a Single Asset	Throughput For a Single Asset	Team Members	Combined Throughput	Process Throughput	Team Utilization
Concept	3	0.333	1	0.333	0.333	100%
Hi-Poly	5	0.2	3	0.6		55.56%
Lo-Poly	3	0.333	2	0.666		50.00%
Rigging	1	1	1	1		33.33%
Com Anim	8	0.125	4	0.5		66.67%
Face Anim	2	0.5	1	0.5		66.67%
Cine Anim	9	0.111	5	0.555		60%
QA	5	0.2	2	0.4		83.33%

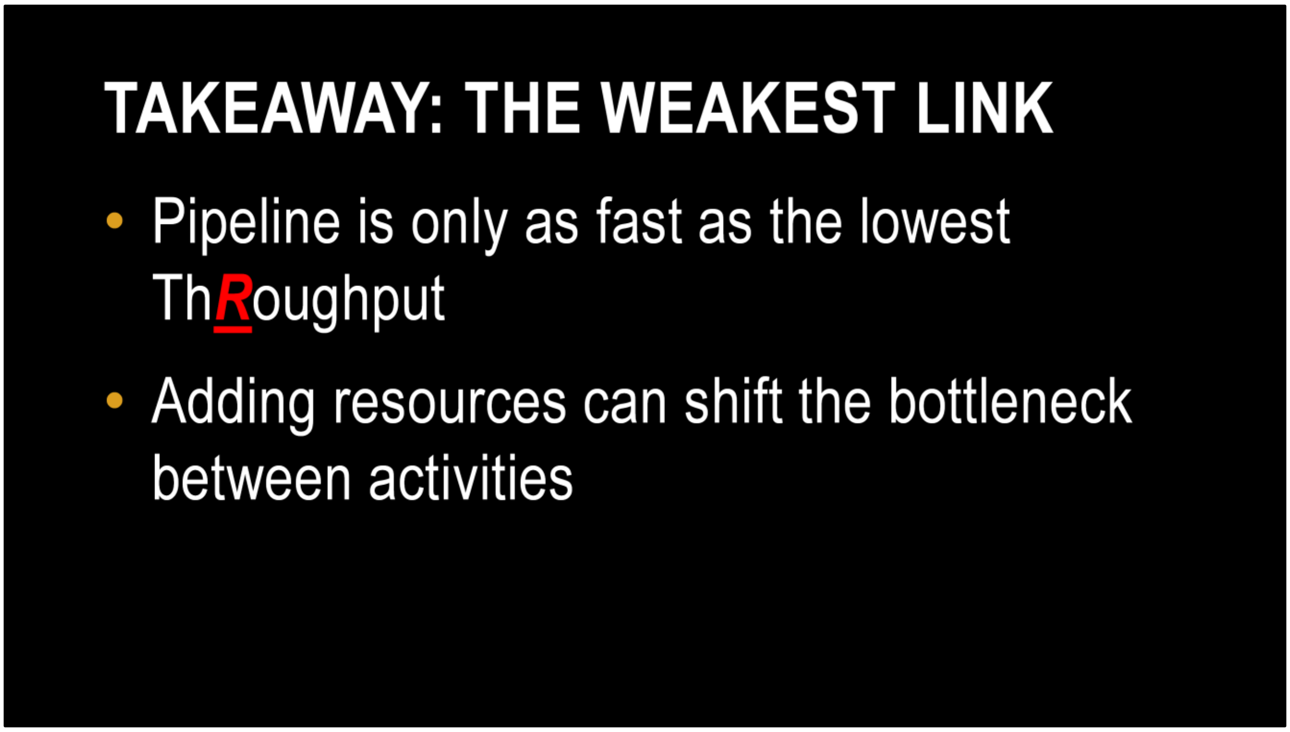
And if we add yet another animator, throughput is still locked at 1/3 of a character per day. All that the additional resource accomplished was creating spare capacity for the animators, which is not the most efficient use of money.



BUT, IF WE ADD ONE TO CONCEPT ART

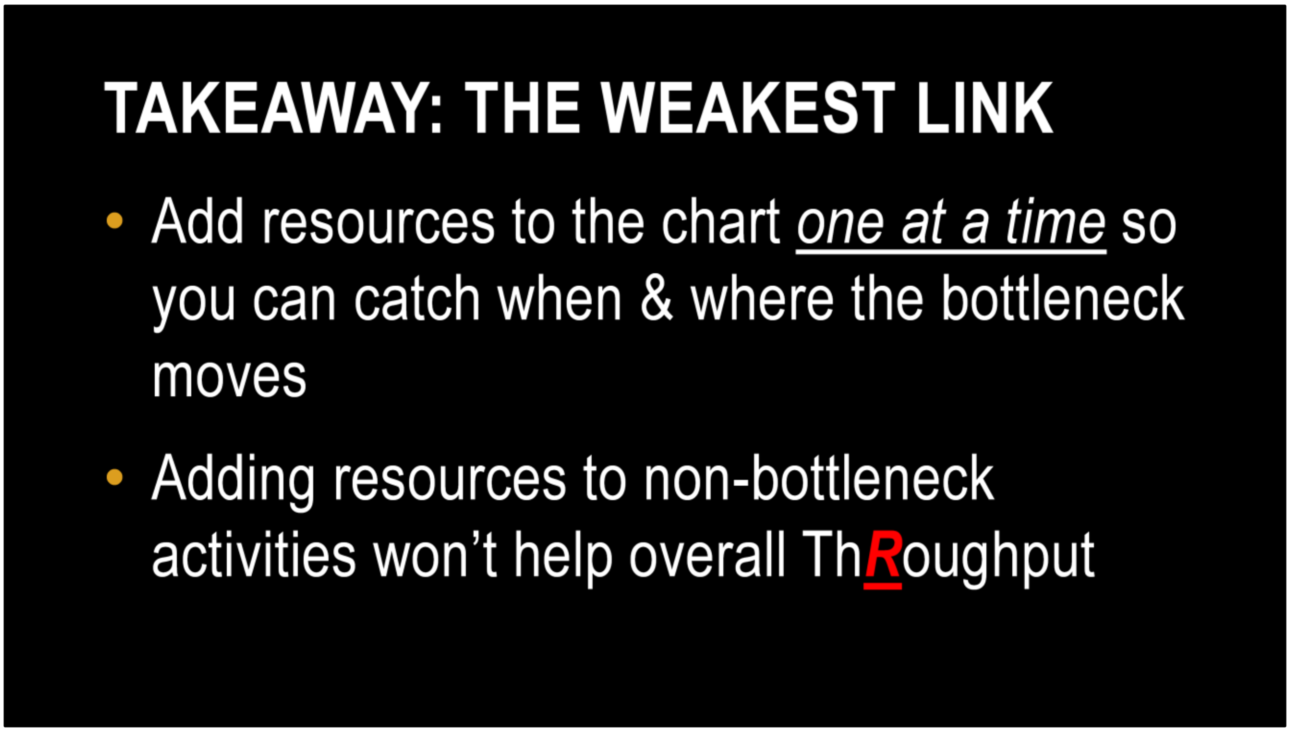
Activity	Flow Time For a Single Asset	Throughput For a Single Asset	Team Members	Combined Throughput	Process Throughput	Team Utilization
Concept	3	0.333	2	0.666	0.400	60%
Hi-Poly	5	0.2	3	0.6		60.67%
Lo-Poly	3	0.333	2	0.666		60.00%
Rigging	1	1	1	1		40.00%
Com Anim	8	0.125	4	0.5		80.00%
Face Anim	2	0.5	1	0.5		80.00%
Cine Anim	9	0.111	5	0.555		72.00%
QA	5	0.2	2	0.4		100.00%

But, if we add a concept artist:
1) The bottleneck now shifts to QA
2) Process capacity increases to 0.4 characters/day
3) The concept artists and cinematic animators have spare capacity



TAKEAWAY: THE WEAKEST LINK

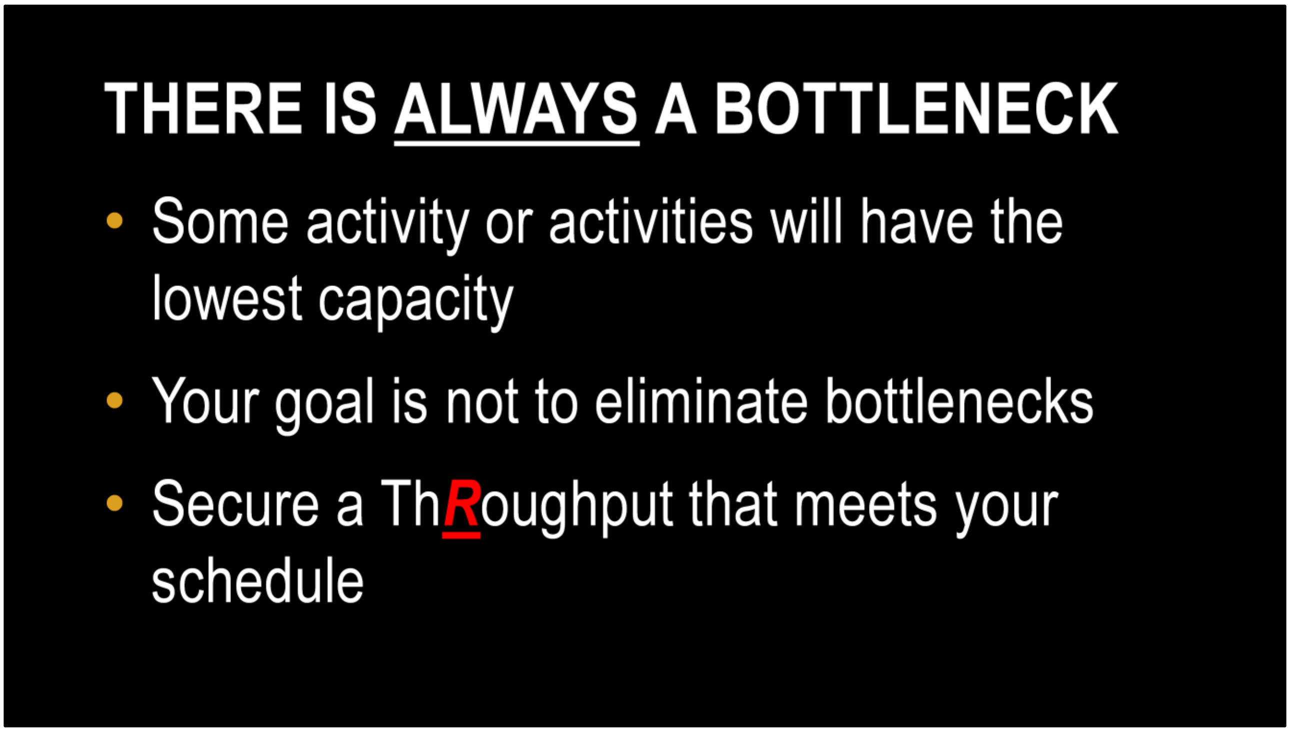
- Pipeline is only as fast as the lowest Th*R*oughput
- Adding resources can shift the bottleneck between activities



TAKEAWAY: THE WEAKEST LINK

- Add resources to the chart *one at a time* so you can catch when & where the bottleneck moves
- Adding resources to non-bottleneck activities won't help overall Th***R***oughput

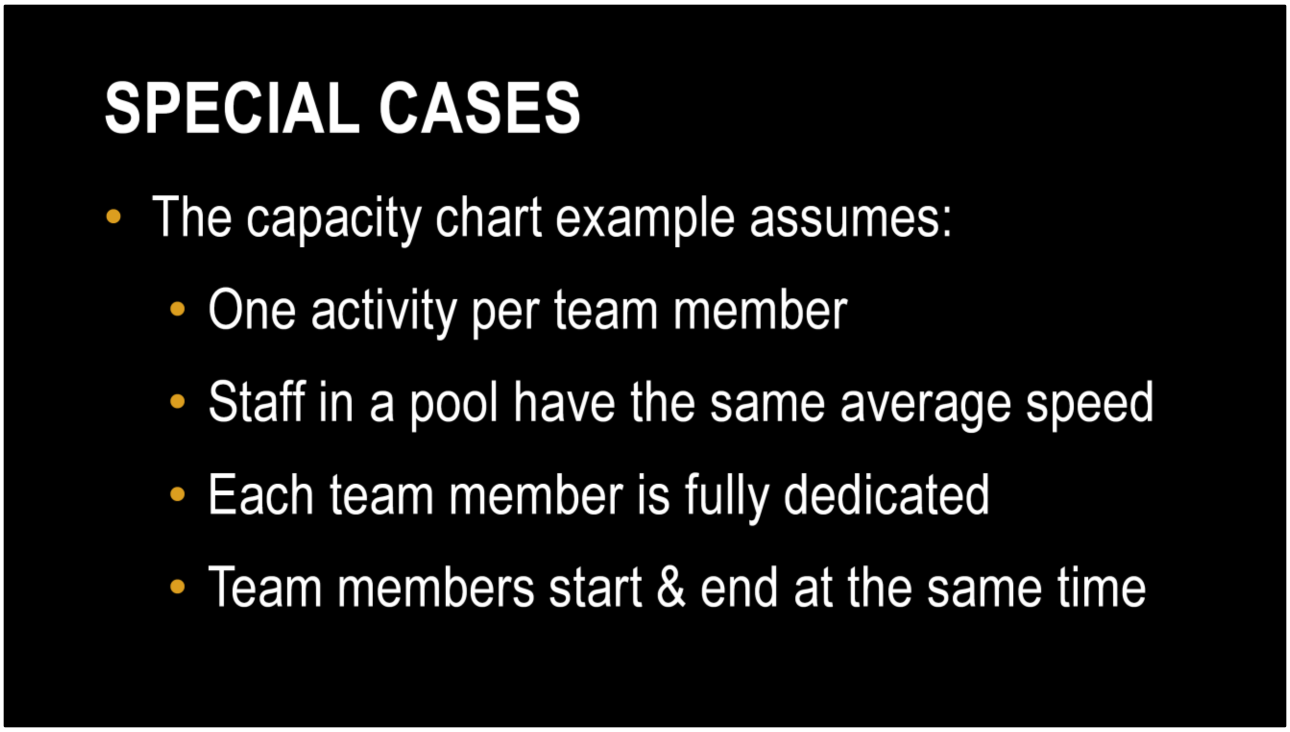
If we had all three cinematic animators at once, we would have seen that the bottleneck moved, but we wouldn't have know when it move. Thus we wouldn't have known which additional team member/s only created spare capacity.



THERE IS ALWAYS A BOTTLENECK

- Some activity or activities will have the lowest capacity
- Your goal is not to eliminate bottlenecks
- Secure a Th***R***oughput that meets your schedule

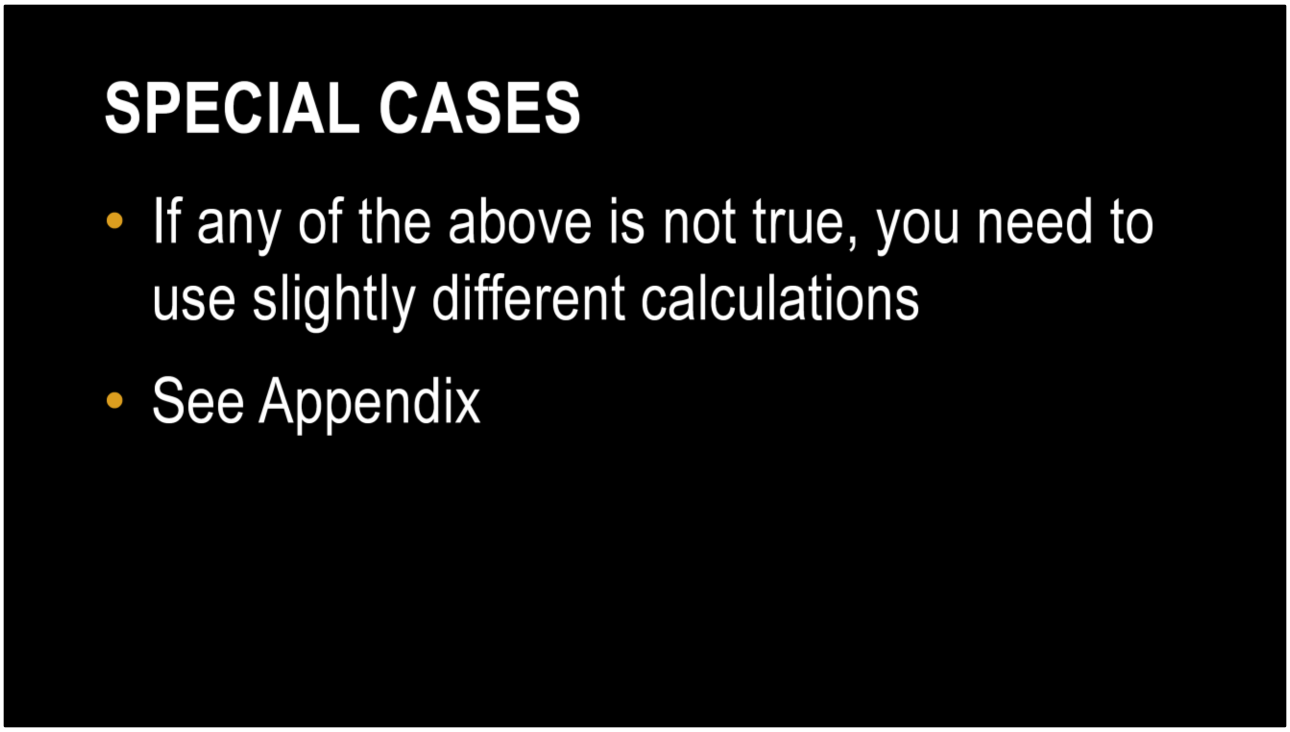
It’s impossible to eliminate all bottlenecks. There will always be a slowest resource.



SPECIAL CASES

- The capacity chart example assumes:
  - One activity per team member
  - Staff in a pool have the same average speed
  - Each team member is fully dedicated
  - Team members start & end at the same time

IE, the example takes a very clean, simplified view of the world.

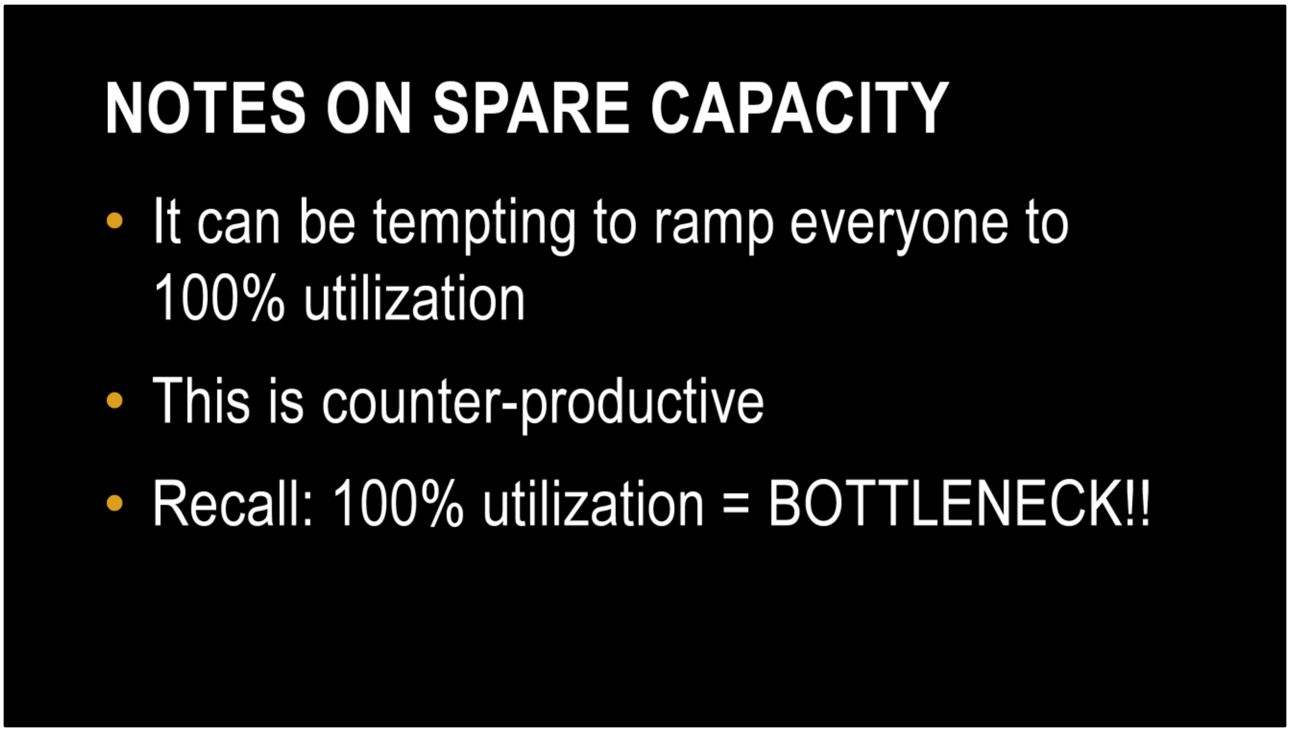


SPECIAL CASES

- If any of the above is not true, you need to use slightly different calculations
- See Appendix

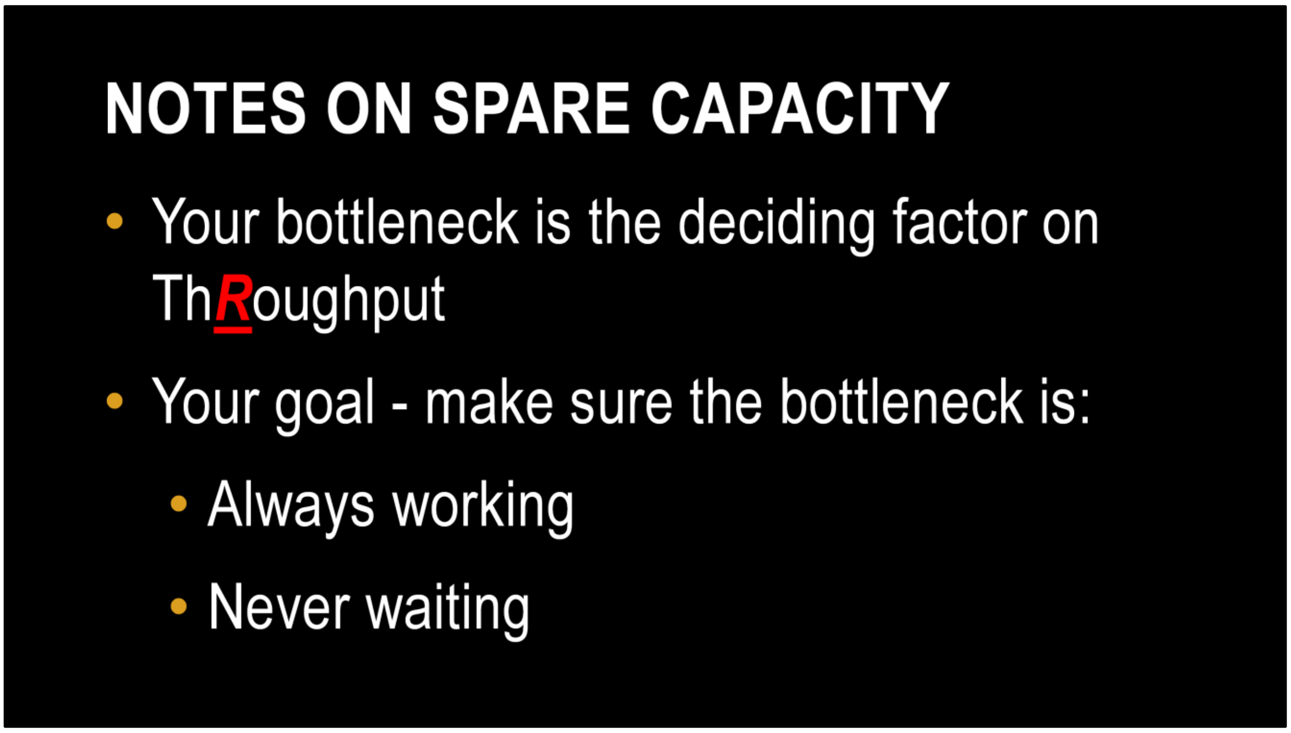
The modified calculations are still just arithmetic. Nothing super-complicated. See Appendix!

NOTES ON SPARE CAPACITY



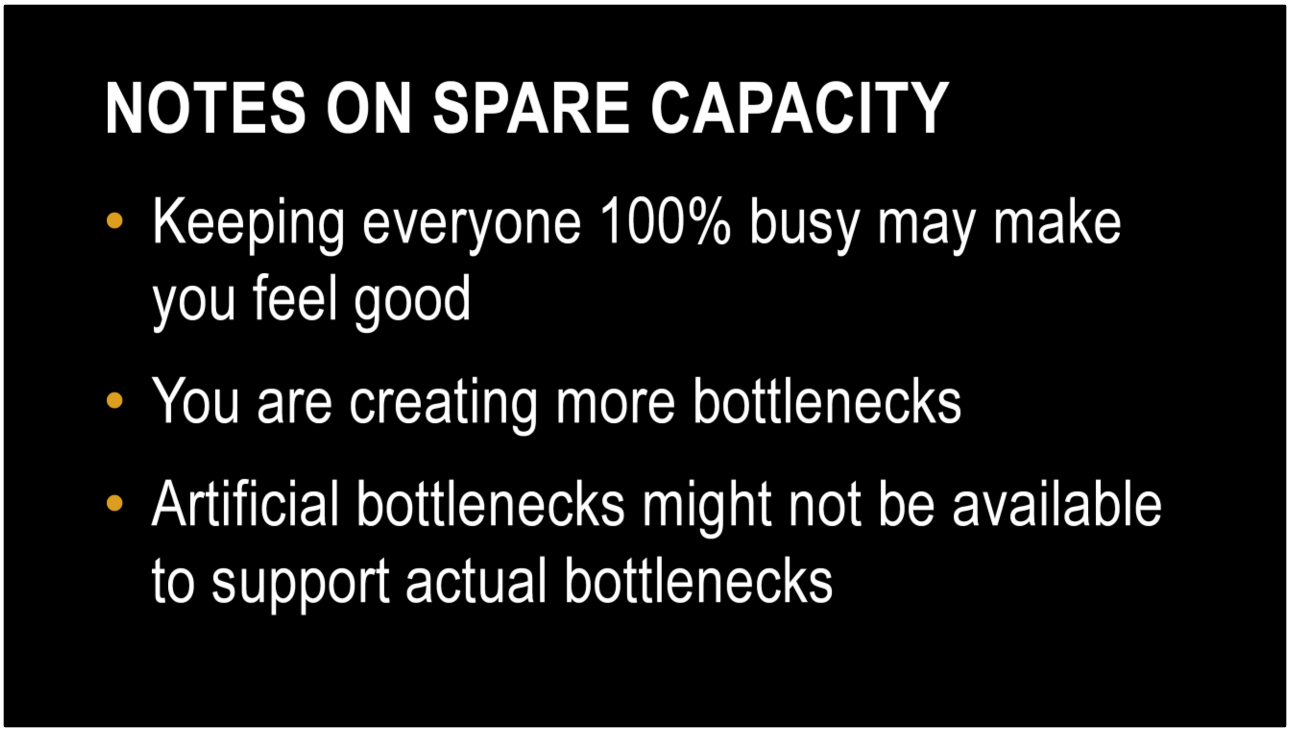
NOTES ON SPARE CAPACITY

- It can be tempting to ramp everyone to 100% utilization
- This is counter-productive
- Recall: 100% utilization = BOTTLENECK!!



NOTES ON SPARE CAPACITY

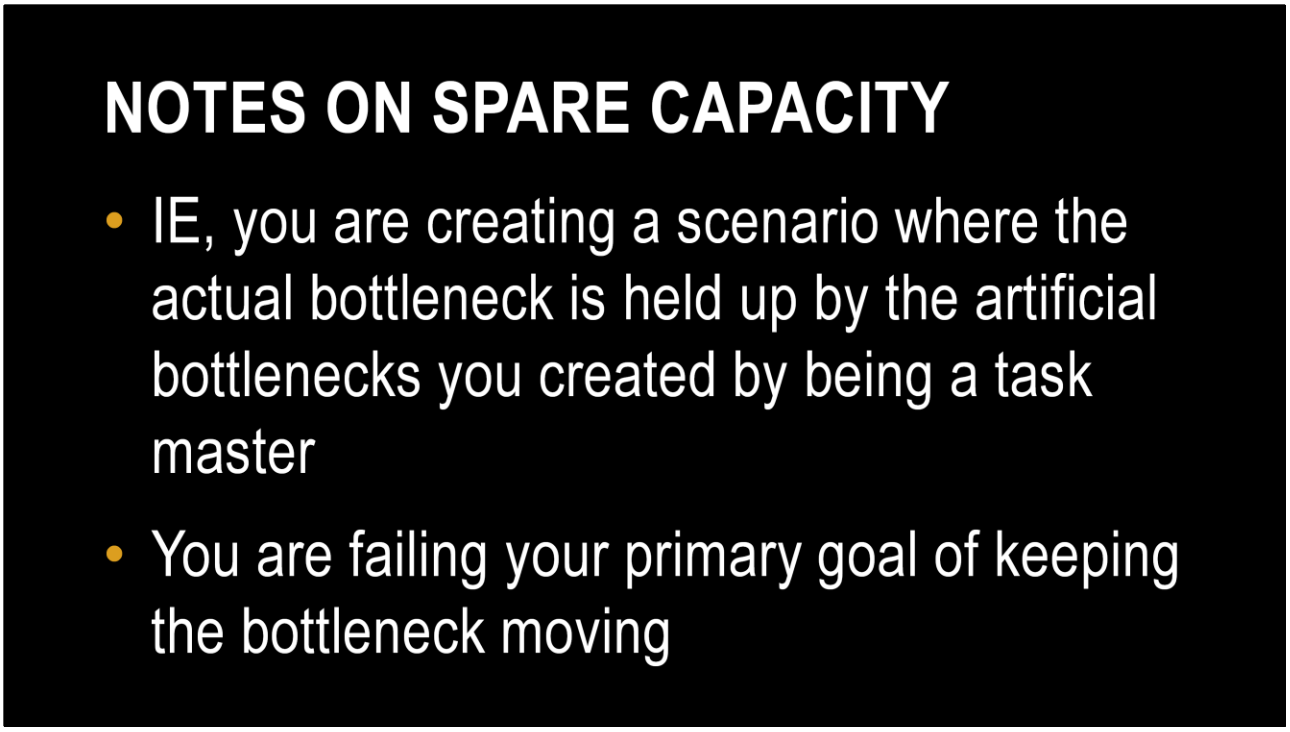
- Your bottleneck is the deciding factor on Th*R*oughput
- Your goal - make sure the bottleneck is:
  - Always working
  - Never waiting



NOTES ON SPARE CAPACITY

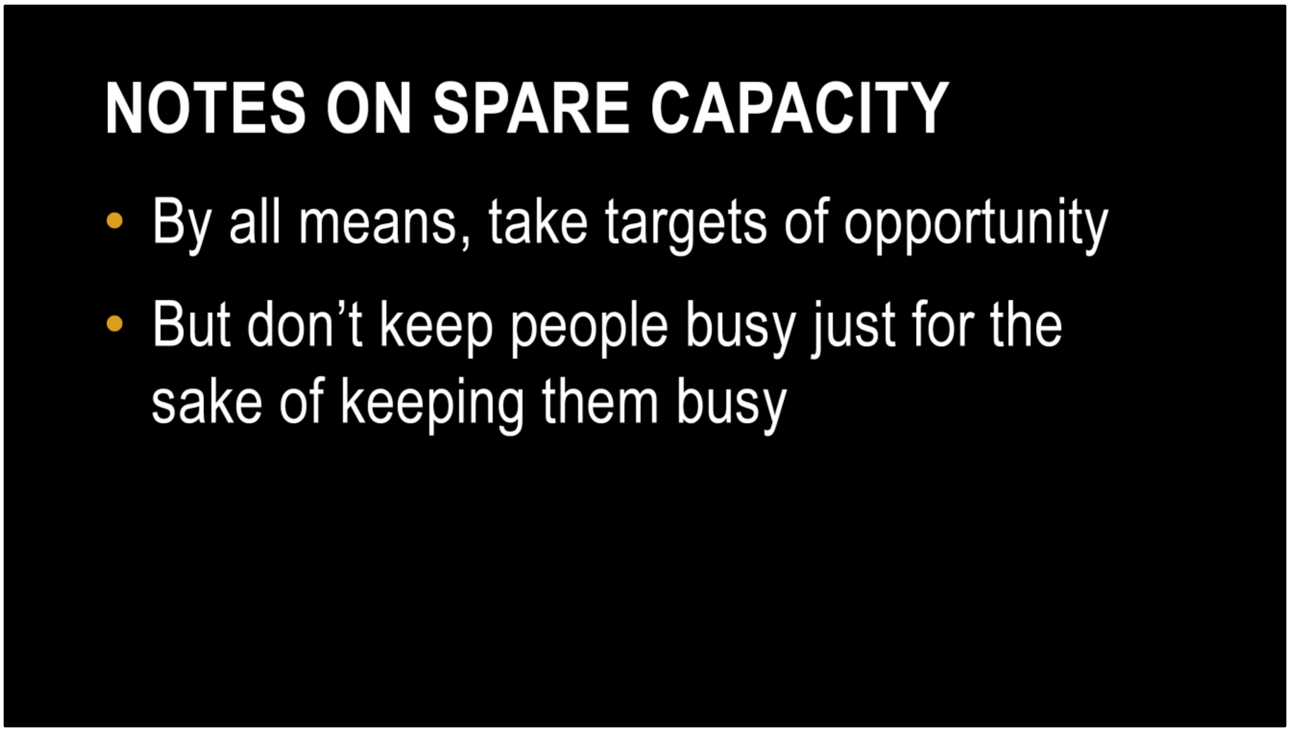
- Keeping everyone 100% busy may make you feel good
- You are creating more bottlenecks
- Artificial bottlenecks might not be available to support actual bottlenecks

- Example: if the actual bottleneck (example: the cinematic animators) runs out of work, and upstream processes (example: the riggers) can’t pass it more work because they’re tied up with busy-work, the bottleneck will stop moving. Thus your overall throughput will slow down.



NOTES ON SPARE CAPACITY

- IE, you are creating a scenario where the actual bottleneck is held up by the artificial bottlenecks you created by being a task master
- You are failing your primary goal of keeping the bottleneck moving



NOTES ON SPARE CAPACITY

- By all means, take targets of opportunity
- But don’t keep people busy just for the sake of keeping them busy

NOTES ON SPARE CAPACITY

- The best use of spare capacity is to alleviate the bottleneck
- If team members with spare capacity can also do bottleneck work, that’s the best place to apply their talents

STORY TIME!

Time for a palate cleanser!

BETWEEN SCYLLA & CHARYBDIS

BETWEEN SCYLLA & CHARYBDIS

- Book XII of *The Odyssey*
- Odysseus must sail through the Straight of Messina

Italy
Straight of Messina
Sicily

BETWEEN SCYLLA & CHARYBDIS

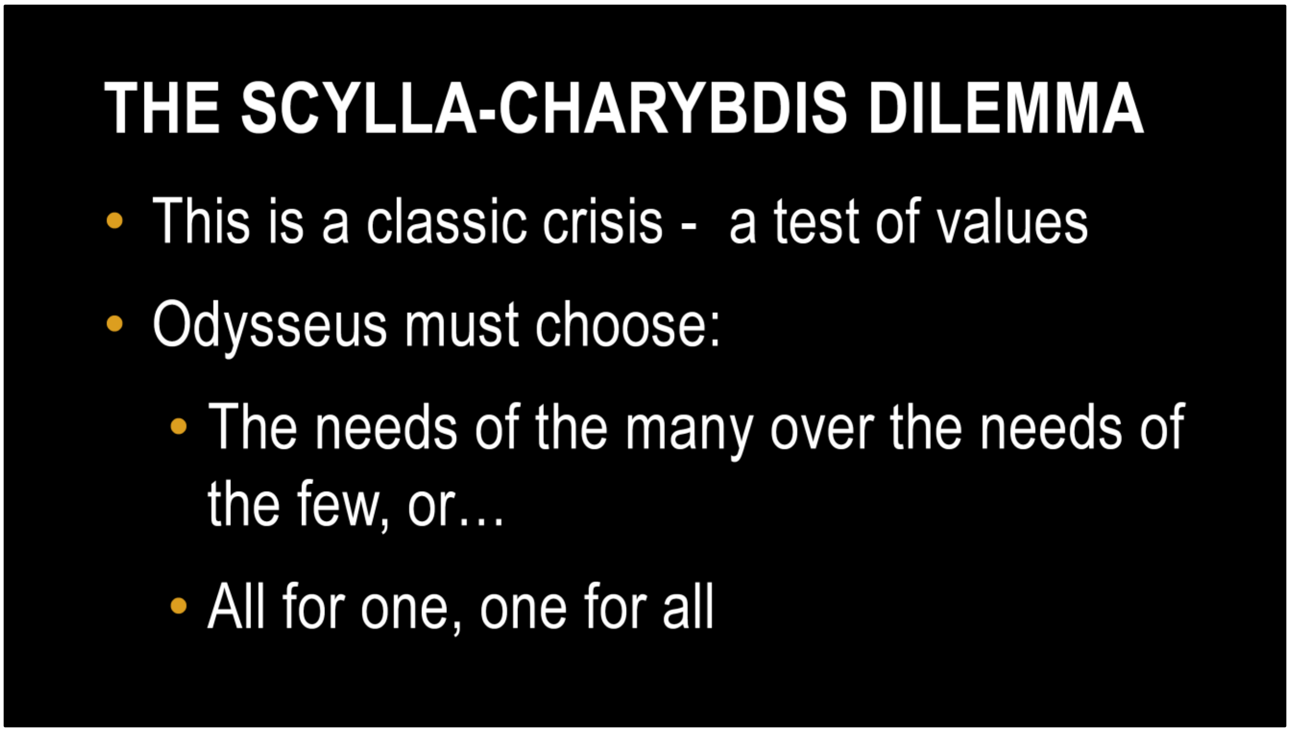
On one side: *Scylla.* On the other: *Charybdis*

Scylla is a six-headed serpent who will devour 6 men from any boat that passes, one with each head. On the other side is Charybdis, and underwater beast who creates whirlpools. She might not catch you, but if she does everyone on the boat dies.

BETWEEN SCYLLA & CHARYBDIS

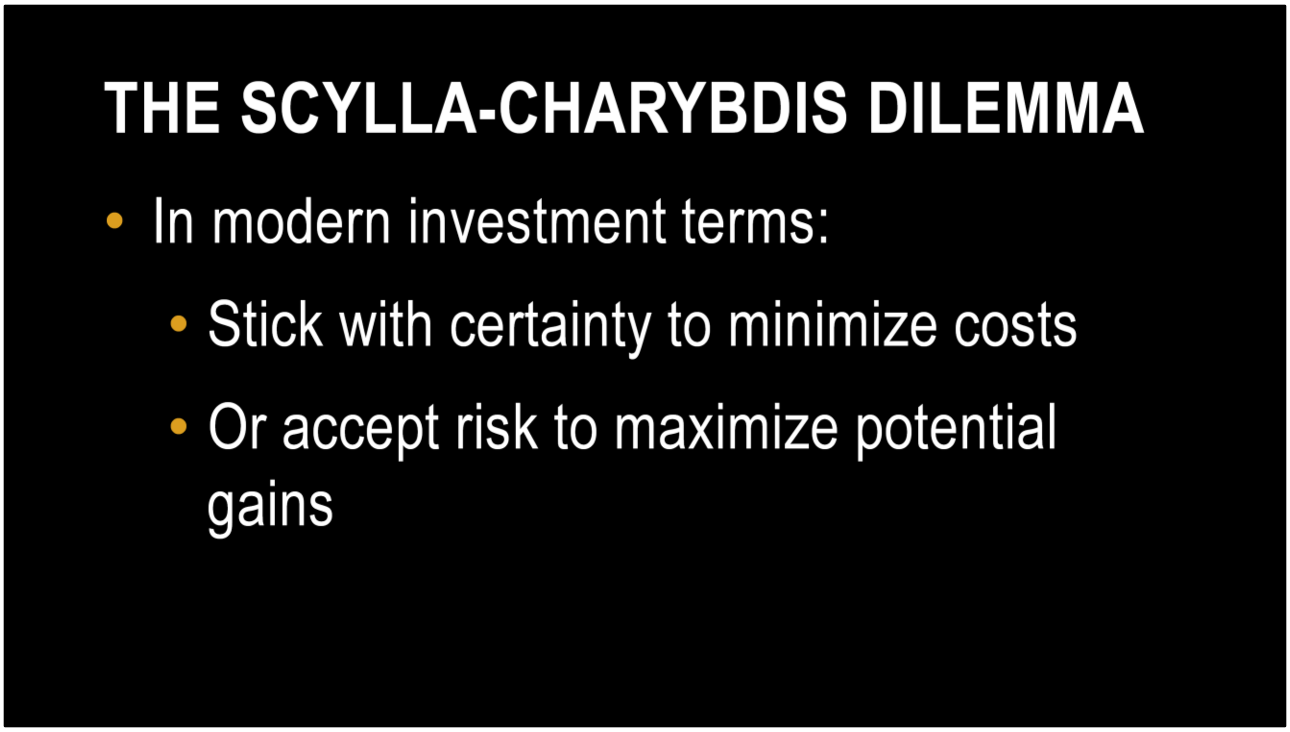
Definitely lose 6 men, or possibly lose ***all*** of them

This is Odysseus’ choice



THE SCYLLA-CHARYBDIS DILEMMA

- This is a classic crisis - a test of values
- Odysseus must choose:
  - The needs of the many over the needs of the few, or…
  - All for one, one for all



THE SCYLLA-CHARYBDIS DILEMMA

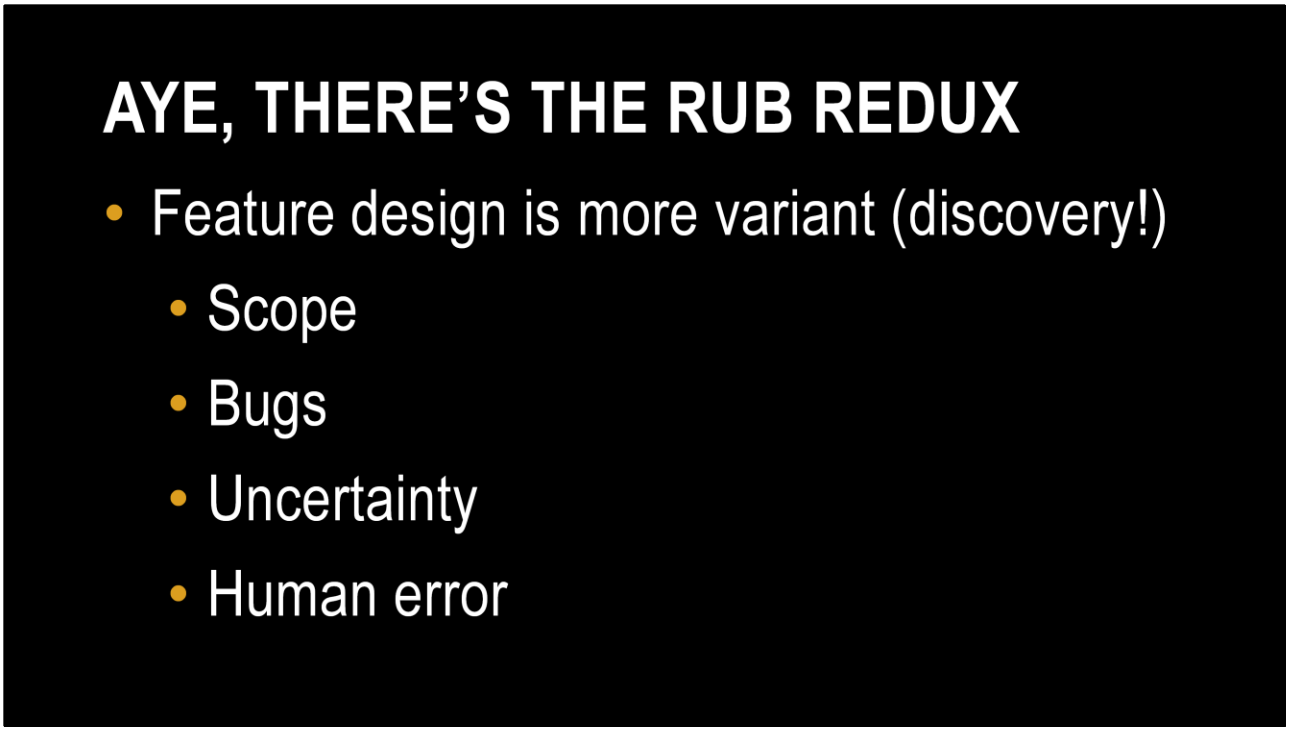
- In modern investment terms:
  - Stick with certainty to minimize costs
  - Or accept risk to maximize potential gains

In the story, Odysseus picked Scylla…but didn’t tell anyone on his boat what would happen.

MOVING BEYOND ART

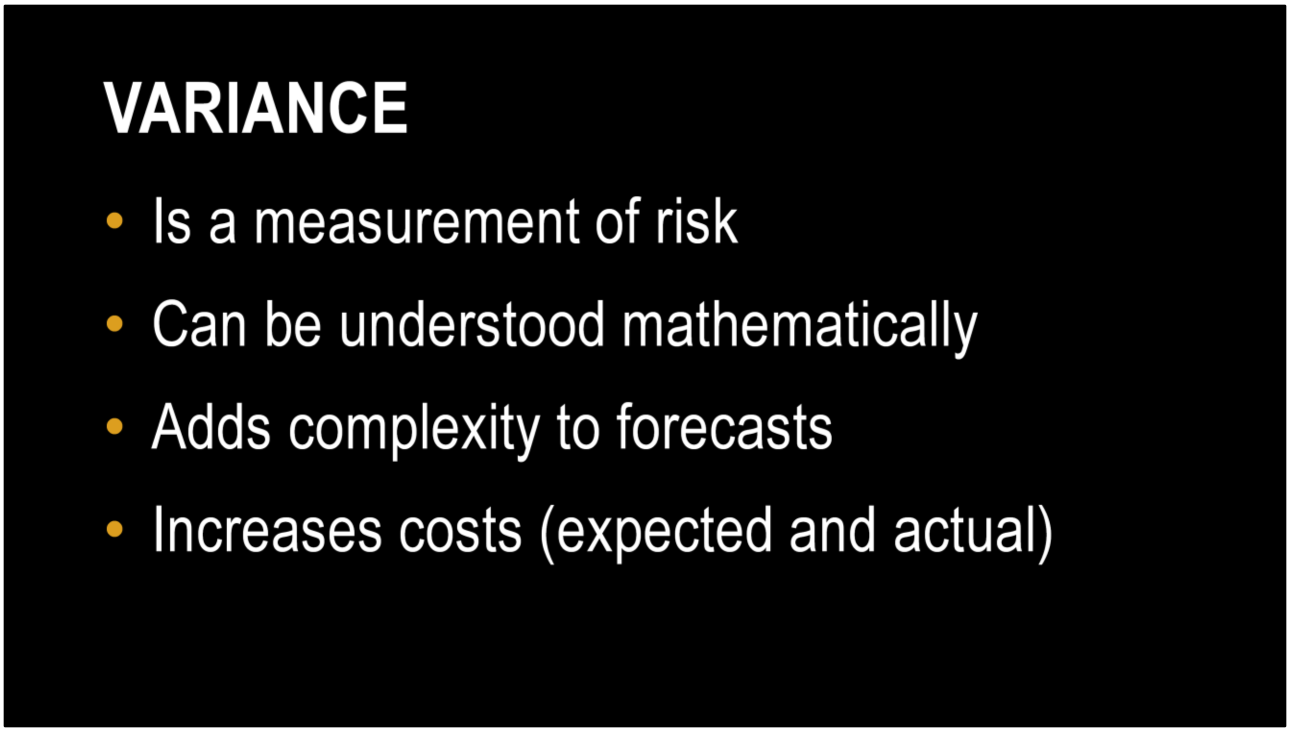
LITTLE'S LAW FOR FEATURES

- Mathematically speaking, Little's Law applies to ***any*** sequential process flow
- This includes feature development
- …with one obvious caveat…



AYE, THERE'S THE RUB REDUX

- Feature design is more variant (discovery!)
  - Scope
  - Bugs
  - Uncertainty
  - Human error



VARIANCE

- Is a measurement of risk
- Can be understood mathematically
- Adds complexity to forecasts
- Increases costs (expected and actual)

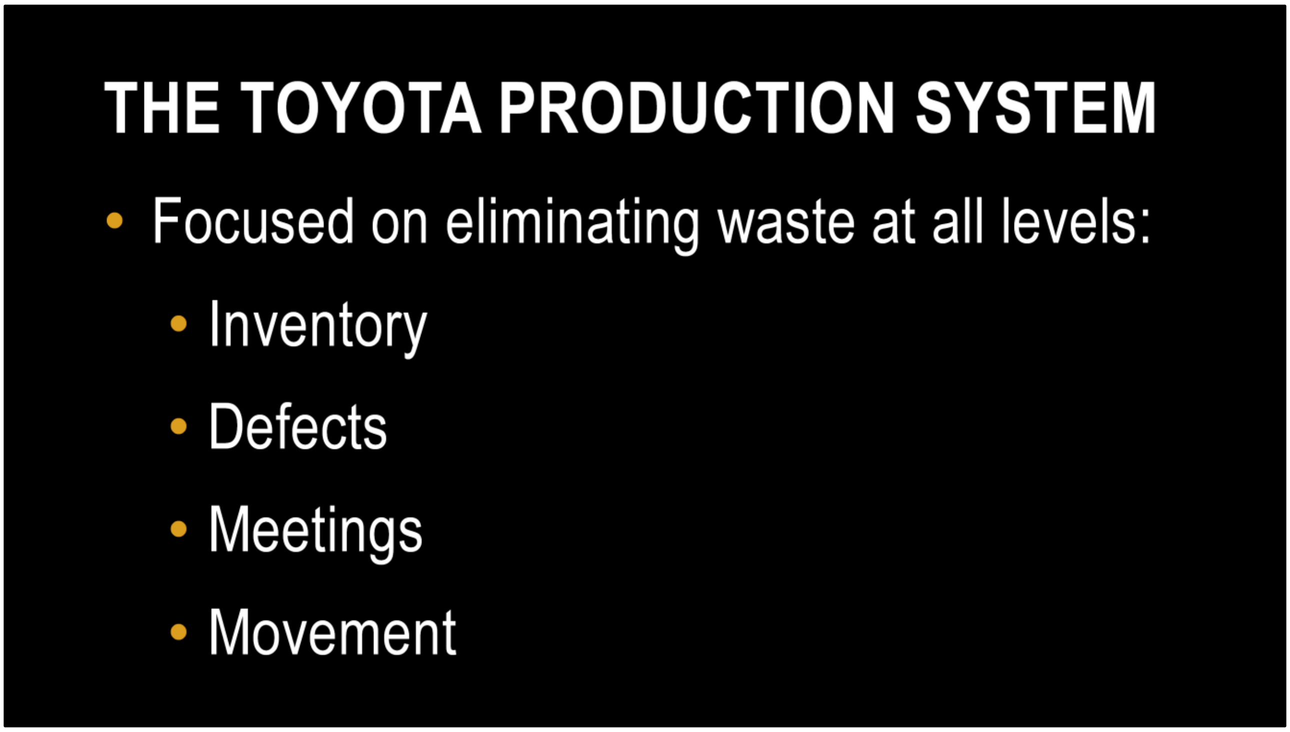
VARIANCE

- Operations science is largely concerned with minimizing the presence and impact of variance
- One path to reducing variance is *eliminating waste*

WHO HAS ELIMINATED PRODUCTION WASTE BETTER THAN ANYBODY ELSE ON THE PLANET?

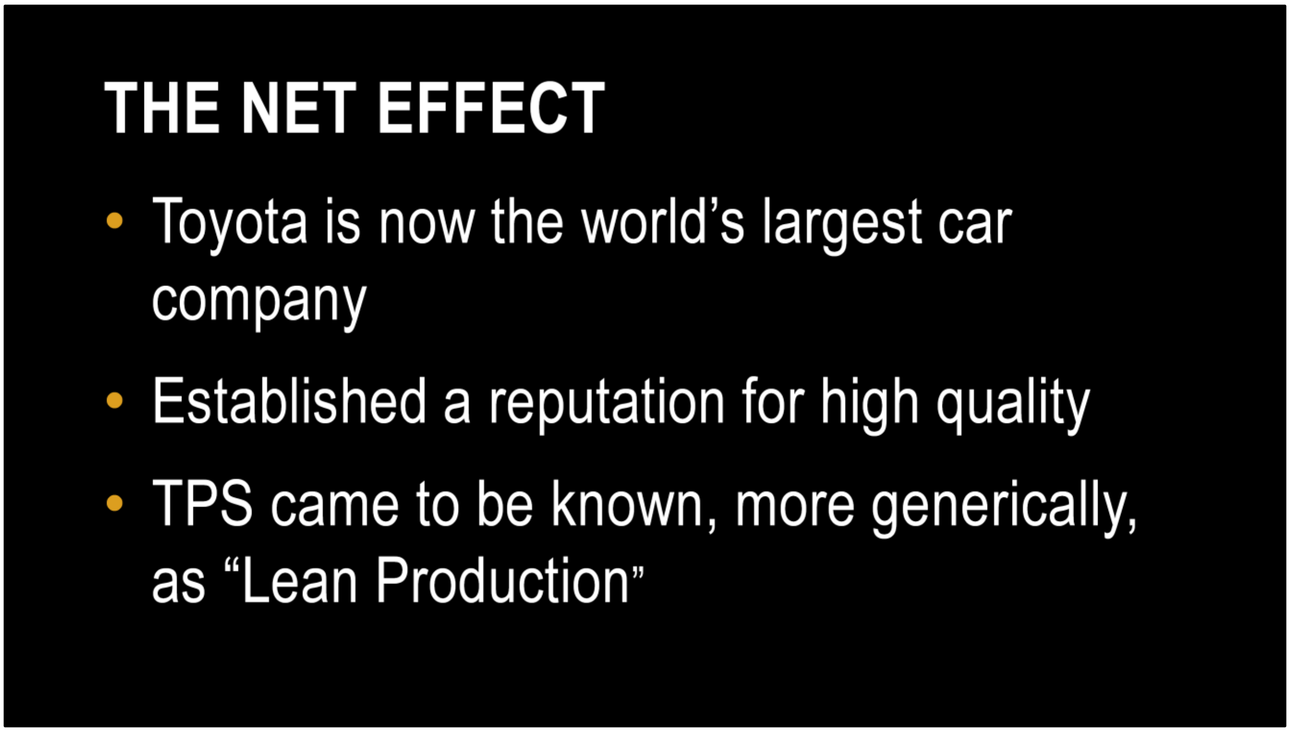
EAST VS. WEST

- 1948
- Japan is in economic ruin
- Toyota is competing with companies from the world's new economic super-power
- How can it possibly succeed?



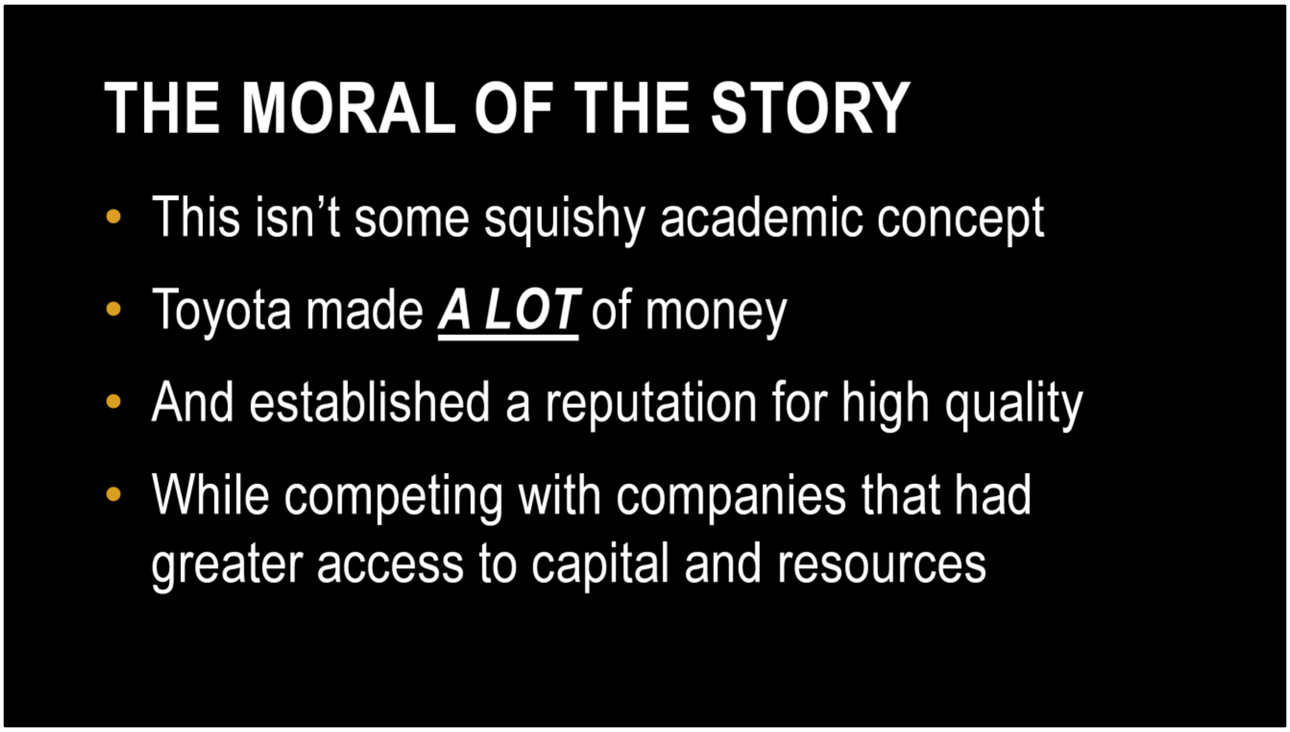
THE TOYOTA PRODUCTION SYSTEM

- Focused on eliminating waste at all levels:
  - Inventory
  - Defects
  - Meetings
  - Movement



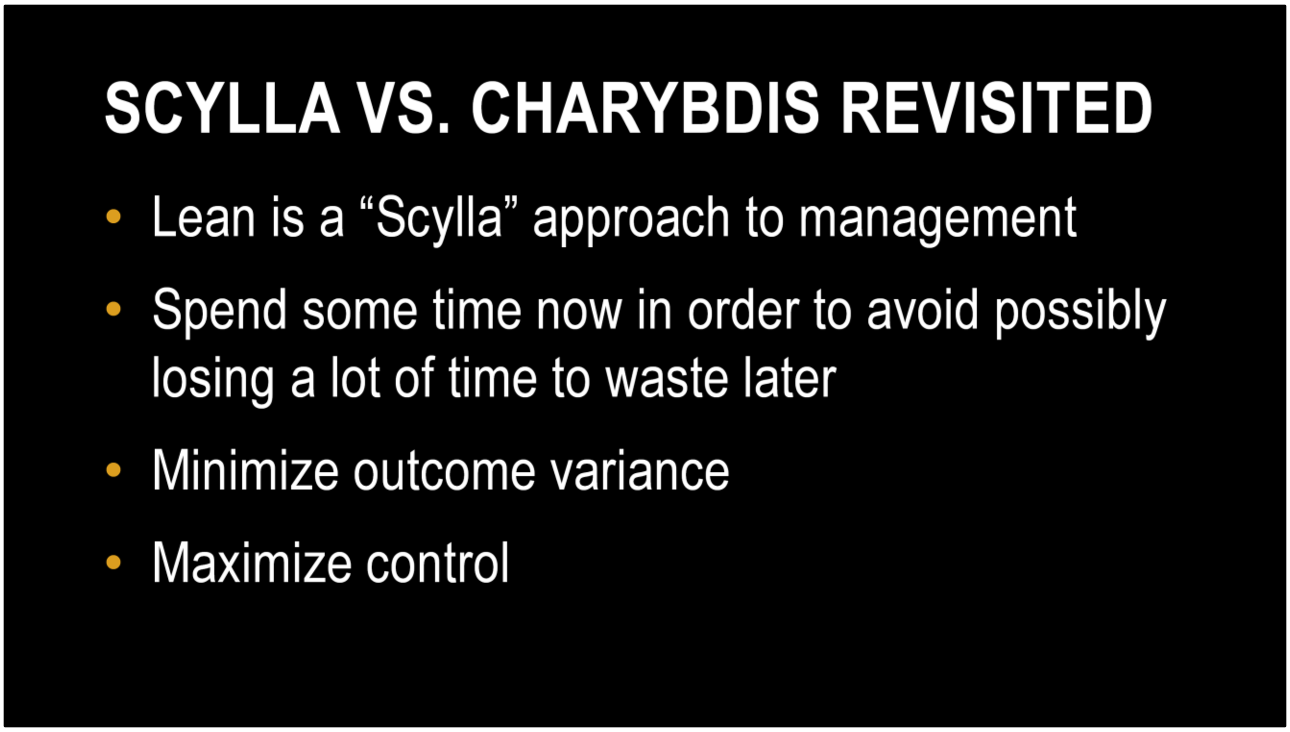
THE NET EFFECT

- Toyota is now the world's largest car company
- Established a reputation for high quality
- TPS came to be known, more generically, as "Lean Production"



THE MORAL OF THE STORY

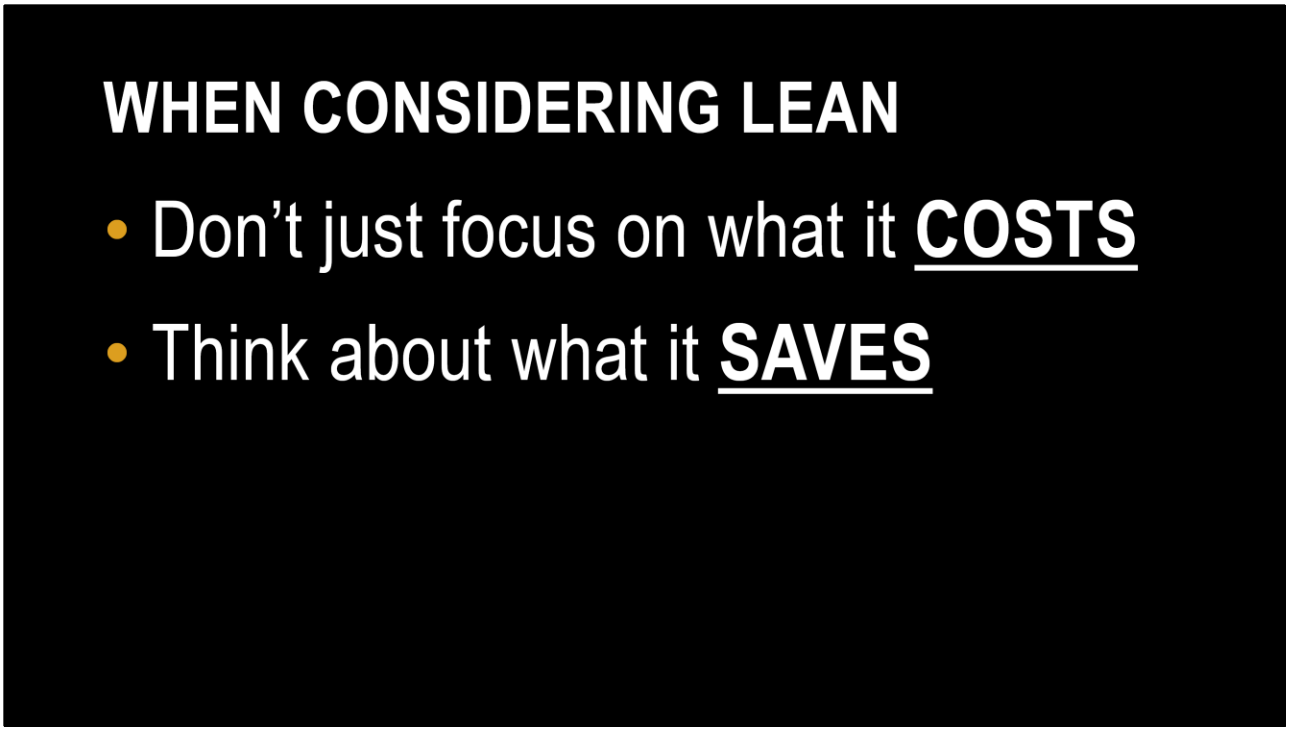
- This isn’t some squishy academic concept
- Toyota made ***A LOT*** of money
- And established a reputation for high quality
- While competing with companies that had greater access to capital and resources



SCYLLA VS. CHARYBDIS REVISITED

- Lean is a “Scylla” approach to management
- Spend some time now in order to avoid possibly losing a lot of time to waste later
- Minimize outcome variance
- Maximize control

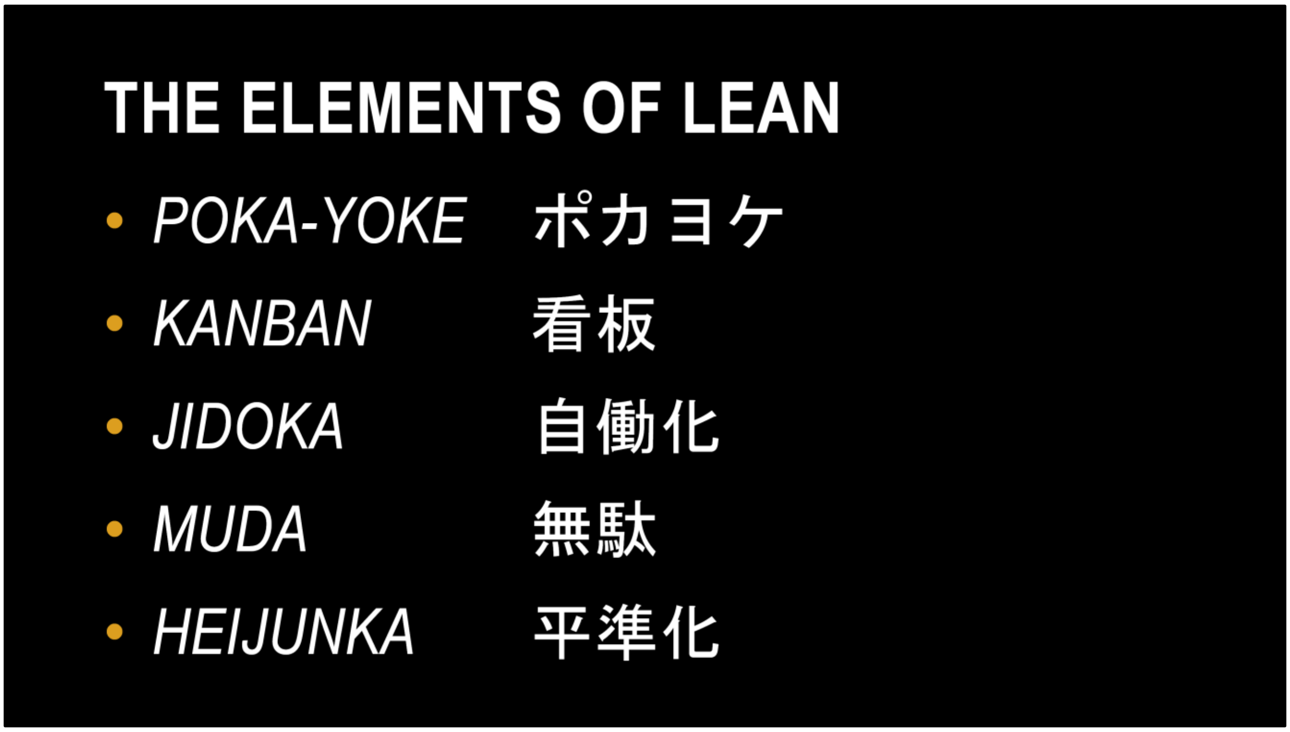
Much like Odysseus, implementing lean means choosing the known cost in order to minimize risk of unknown costs. There was no risk with Scylla: you lose 6 sailors, no more, no less.



WHEN CONSIDERING LEAN

- Don’t just focus on what it **COSTS**
- Think about what it **SAVES**

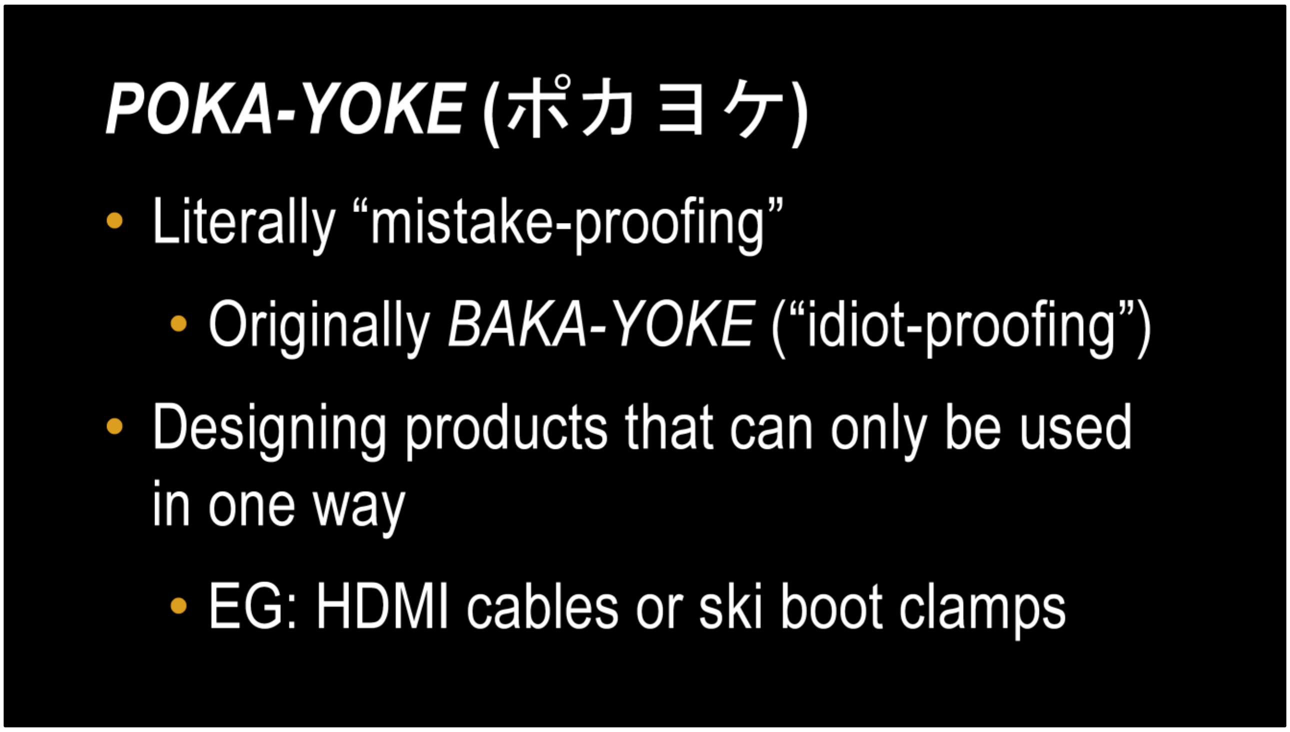
AN OUNCE OF PREVENTION: LEAN PRODUCTION



THE ELEMENTS OF LEAN

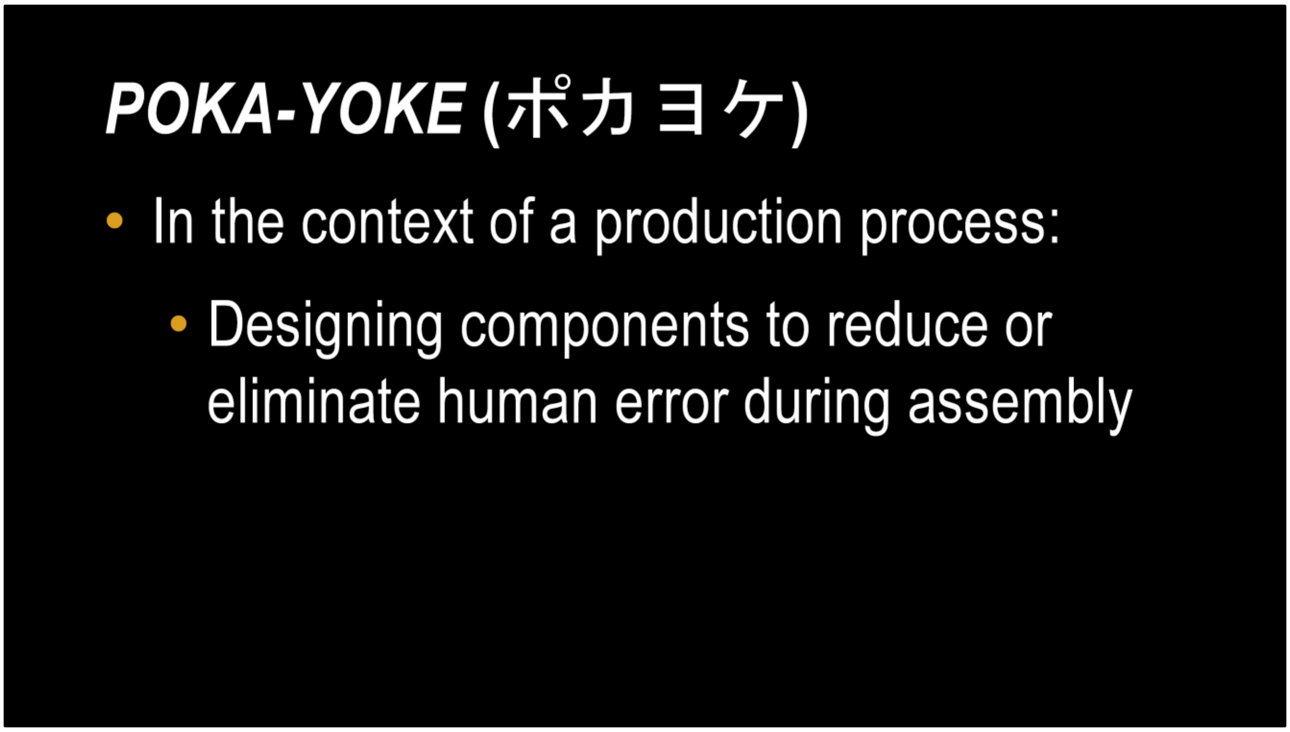
• *POKA-YOKE*	ポカヨケ
• *KANBAN*	看板
• *JIDOKA*	自働化
• *MUDA*	無駄
• *HEIJUNKA*	平準化

POKA-YOKE
ポカヨケ



POKA-YOKE (ポカヨケ)

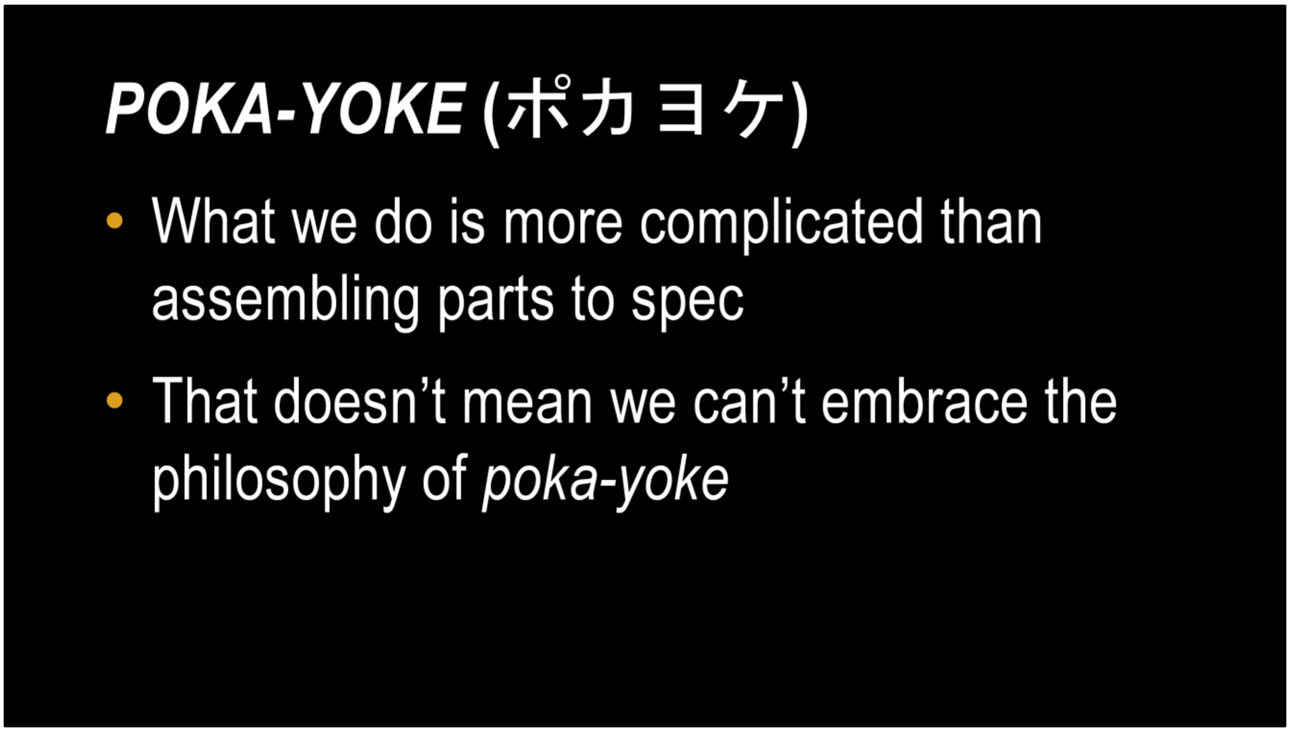
- Literally “mistake-proofing”
  - Originally *BAKA-YOKE* (“idiot-proofing”)
- Designing products that can only be used in one way
  - EG: HDMI cables or ski boot clamps



POKA-YOKE (ポカヨケ)

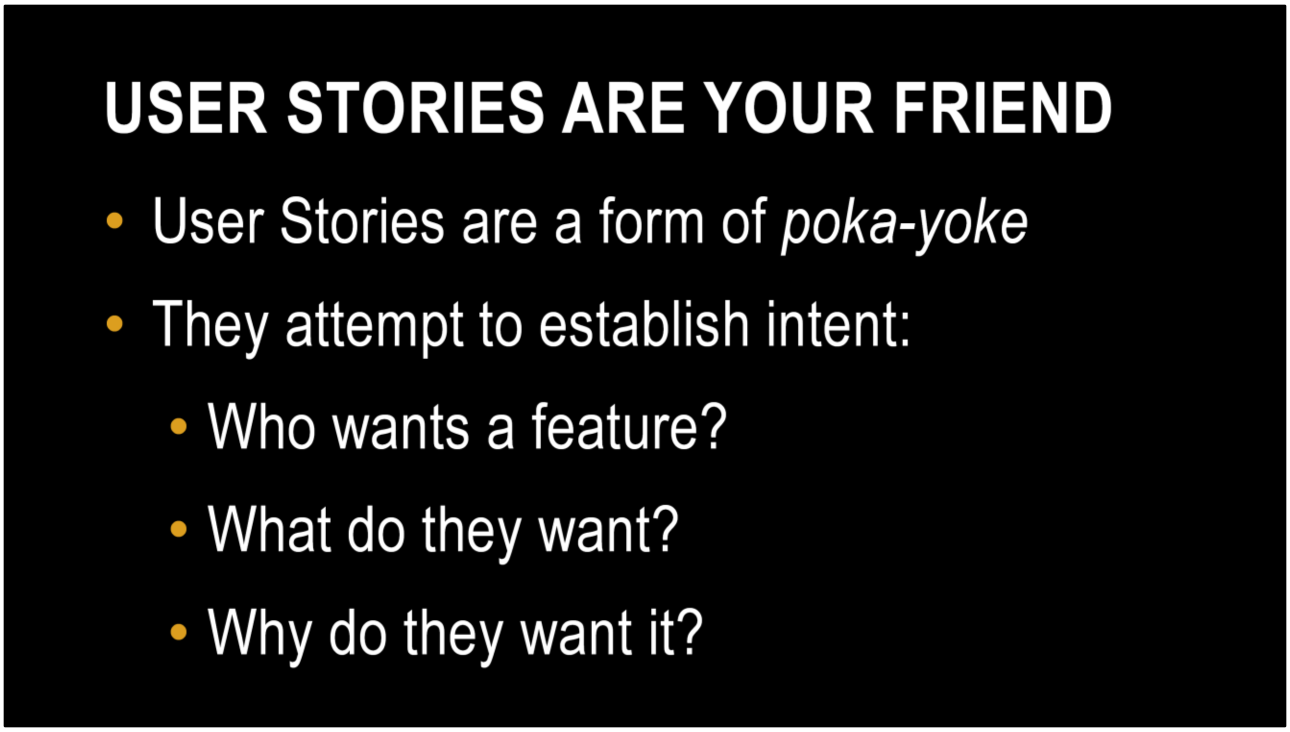
- In the context of a production process:
  - Designing components to reduce or eliminate human error during assembly

Examples: doors can only be mounted one way, bumpers can only be attached one way



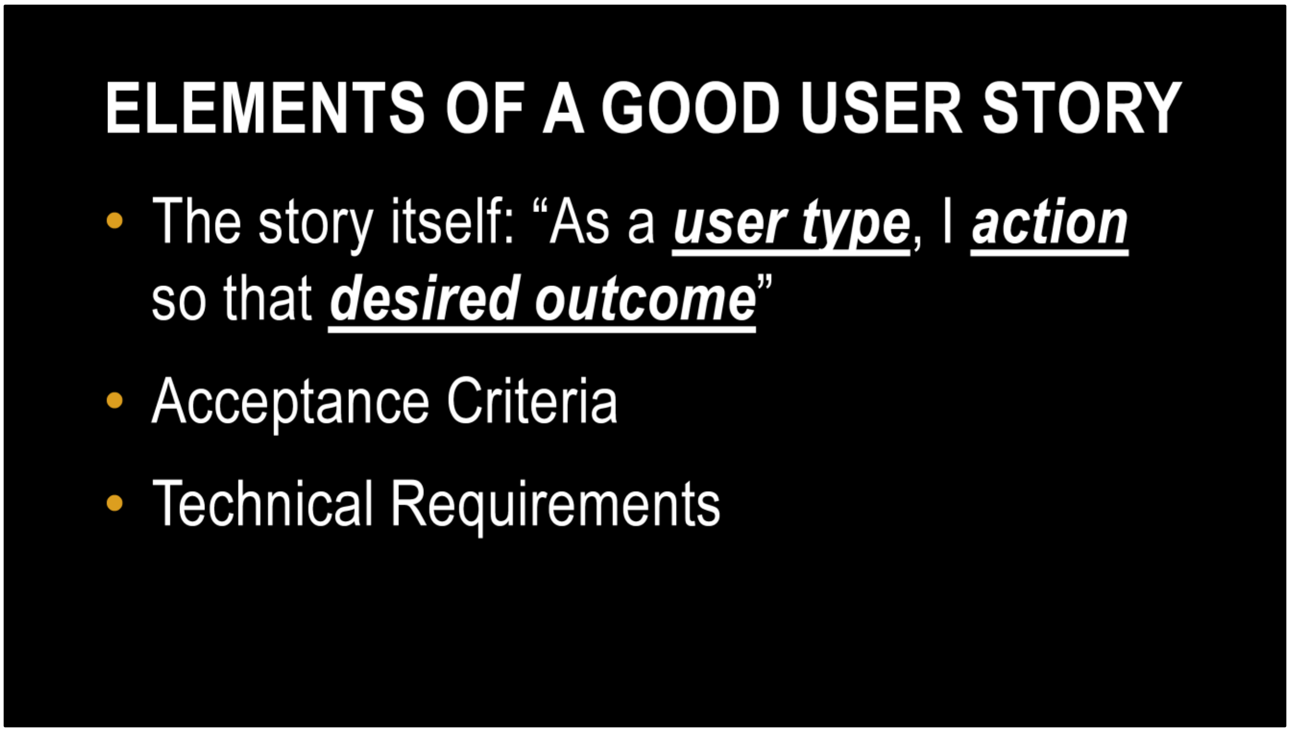
POKA-YOKE (ポカヨケ)

- What we do is more complicated than assembling parts to spec
- That doesn’t mean we can’t embrace the philosophy of *poka-yoke*



USER STORIES ARE YOUR FRIEND

- User Stories are a form of *poka-yoke*
- They attempt to establish intent:
  - Who wants a feature?
  - What do they want?
  - Why do they want it?

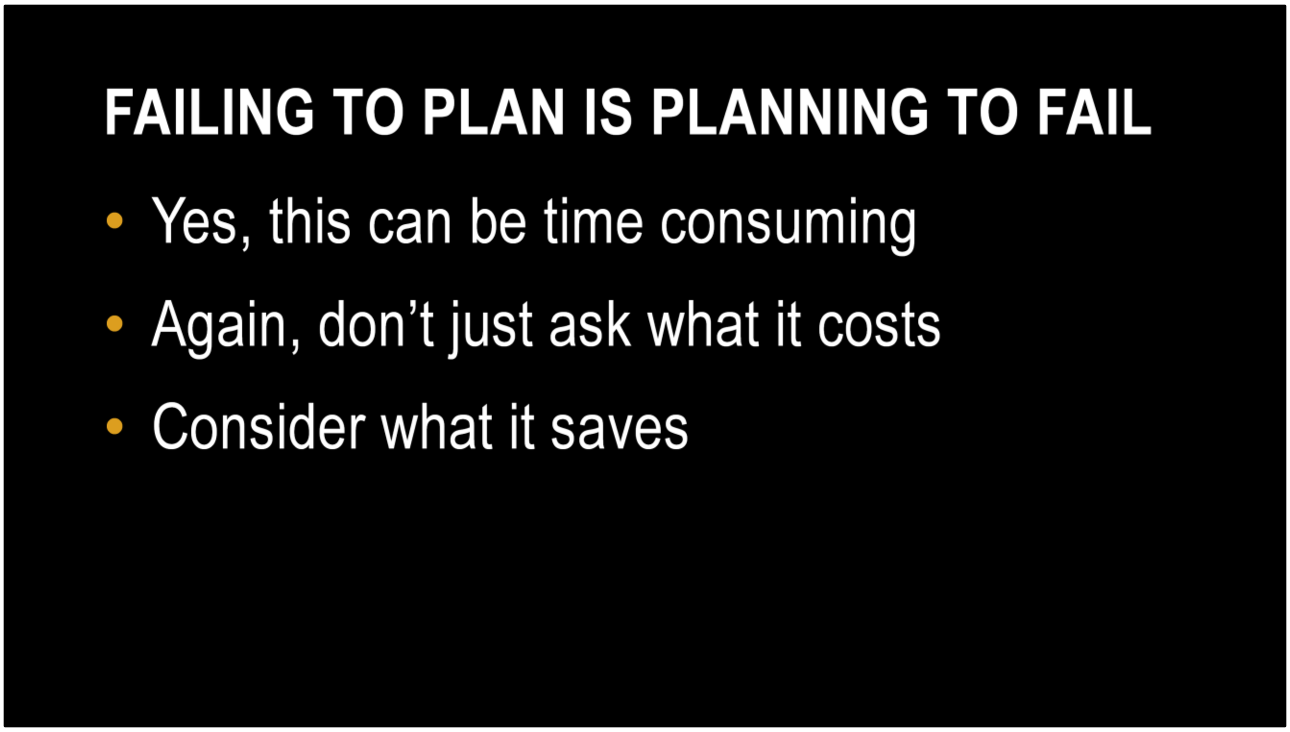


ELEMENTS OF A GOOD USER STORY

- The story itself: "As a ***user type***, I ***action*** so that ***desired outcome***"
- Acceptance Criteria
- Technical Requirements

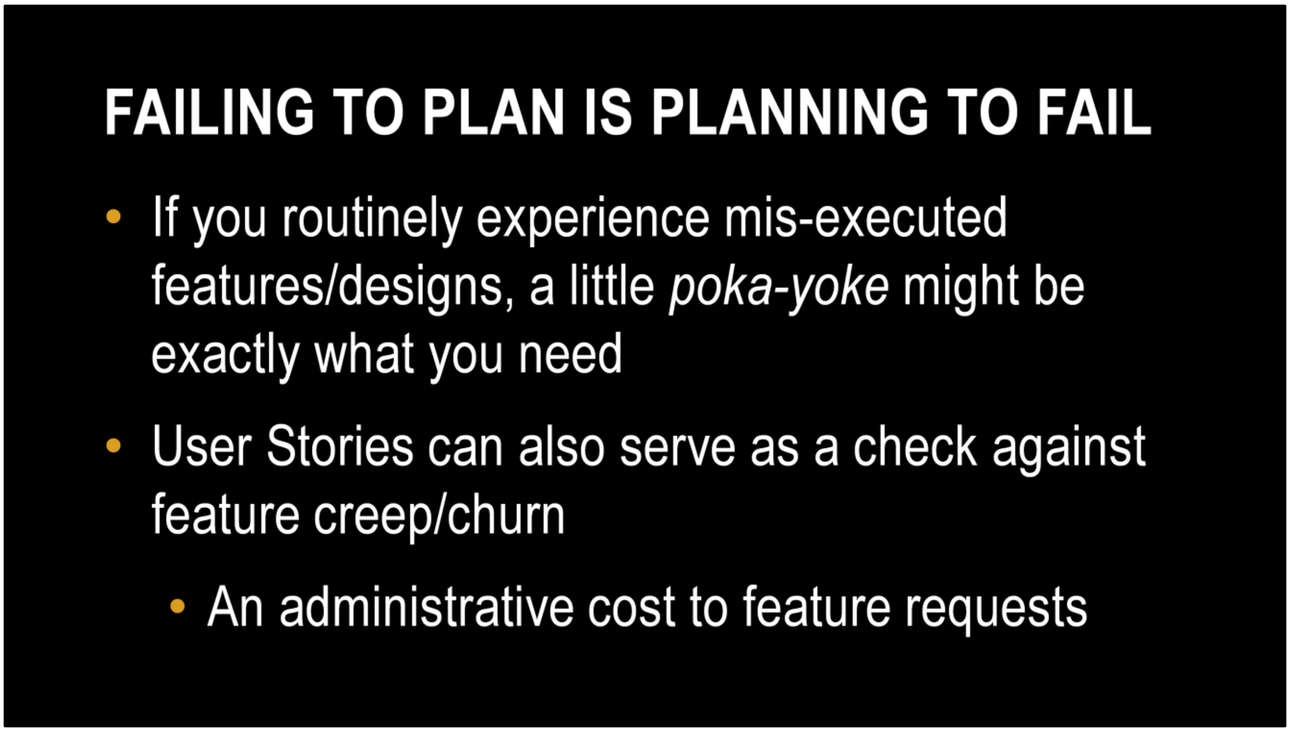
Examples:

- User Stories
  - As a player, I jump, so that I can traverse the environment
  - As an animator, I have an animation blending tool, so I can make a smooth combat experience
  - As an engineer, I have continuous integration, so that I can maintain a smooth and efficient build process
- Acceptance criteria
  - To consider this feature complete:
    - Pushing "A" needs to make the character jump
    - The longer I hold down A, the higher the character jumps
    - I need an easy to access variable to adjust the maximum and minimum heights of the jump
- Technical requirements
  - This code needs to interact with Class X
  - It needs to accept Object Y
  - The code can occupy a maximum of Z bytes in memory
  - It has to accept A input and produce B output



FAILING TO PLAN IS PLANNING TO FAIL

- Yes, this can be time consuming
- Again, don’t just ask what it costs
- Consider what it saves

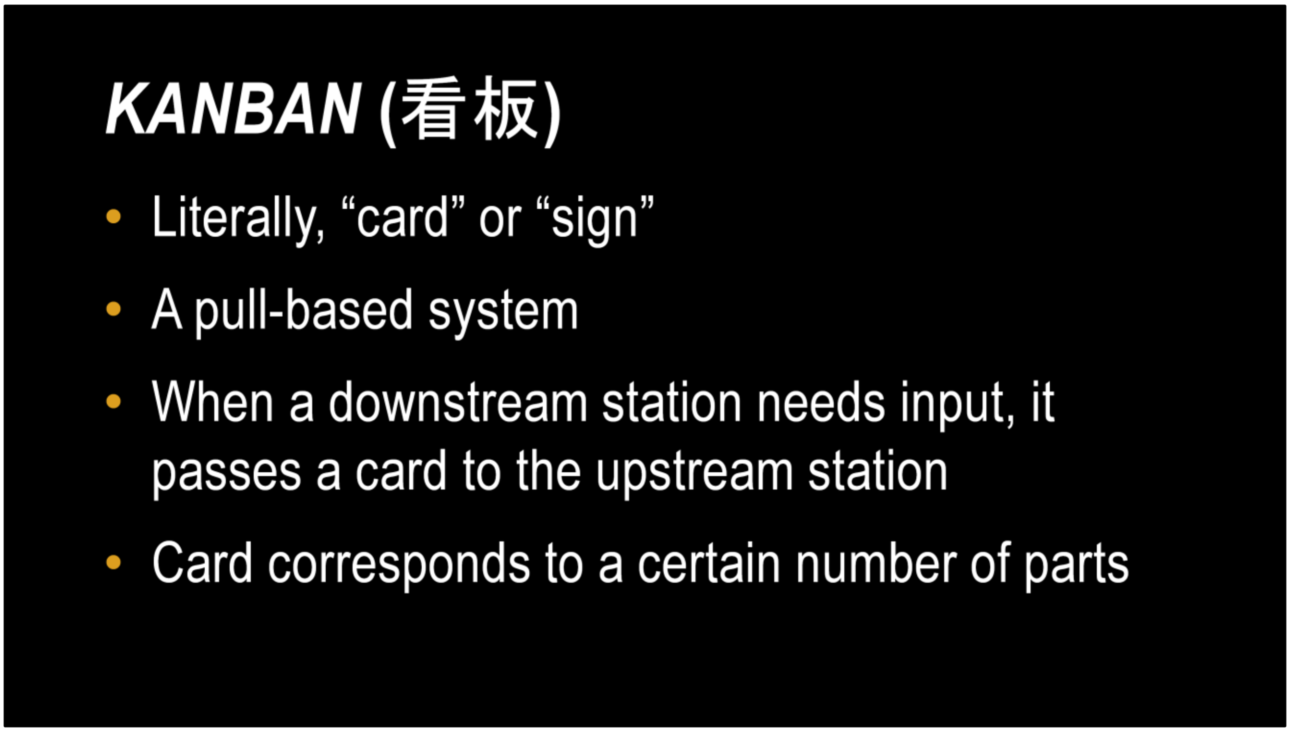


FAILING TO PLAN IS PLANNING TO FAIL

- If you routinely experience mis-executed features/designs, a little *poka-yoke* might be exactly what you need
- User Stories can also serve as a check against feature creep/churn
  - An administrative cost to feature requests

If you have a designer or creative lead who is notorious for making impulse-drive feature request, and thus creating a lot of churn, a little bit of administrative friction can curtail that. He/she has to decide if he/she wants the feature badly enough to write a user story for it.

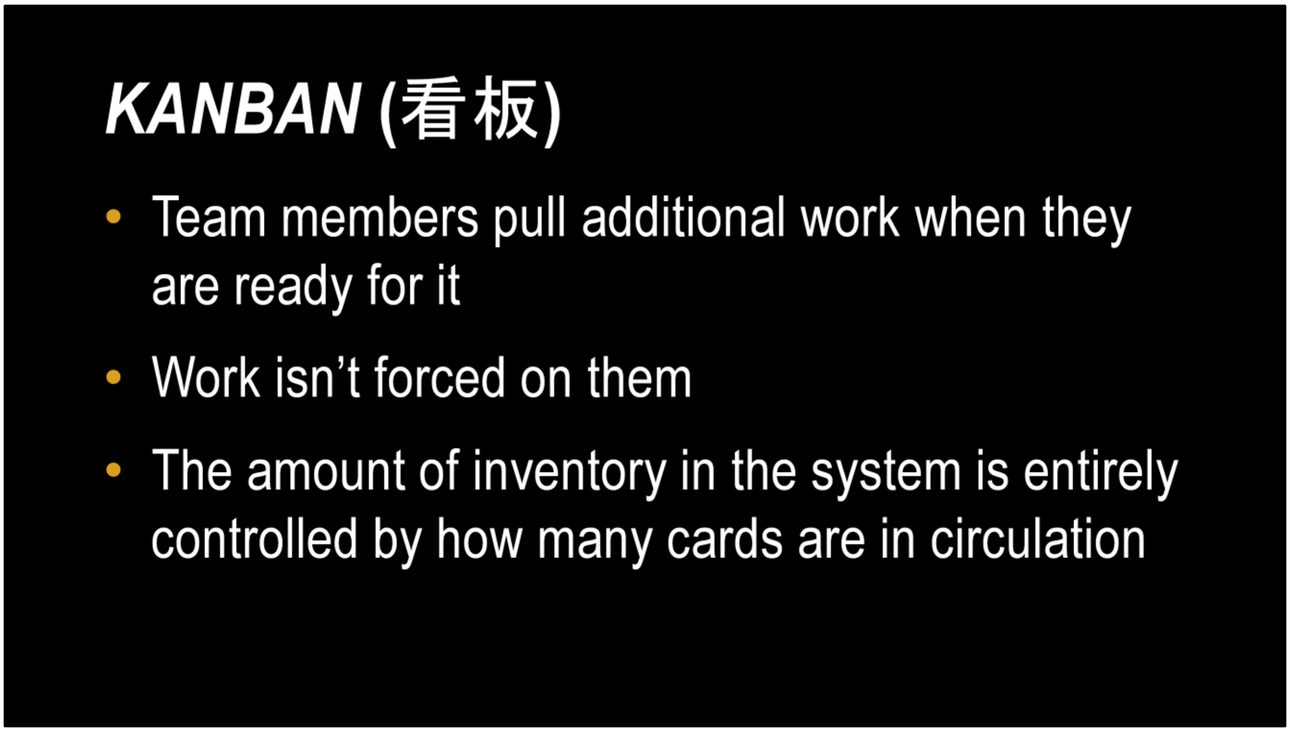
KANBAN
看板



KANBAN (看板)

- Literally, “card” or “sign”
- A pull-based system
- When a downstream station needs input, it passes a card to the upstream station
- Card corresponds to a certain number of parts

Let’s say I’m the guy who mounts doors on cars, and you’re the person who assembles the doors from component parts. When I run out of doors, I put a card in a cart and slide it to you. You put a certain number (dictated by the card) of doors in the cart and send it back to me.



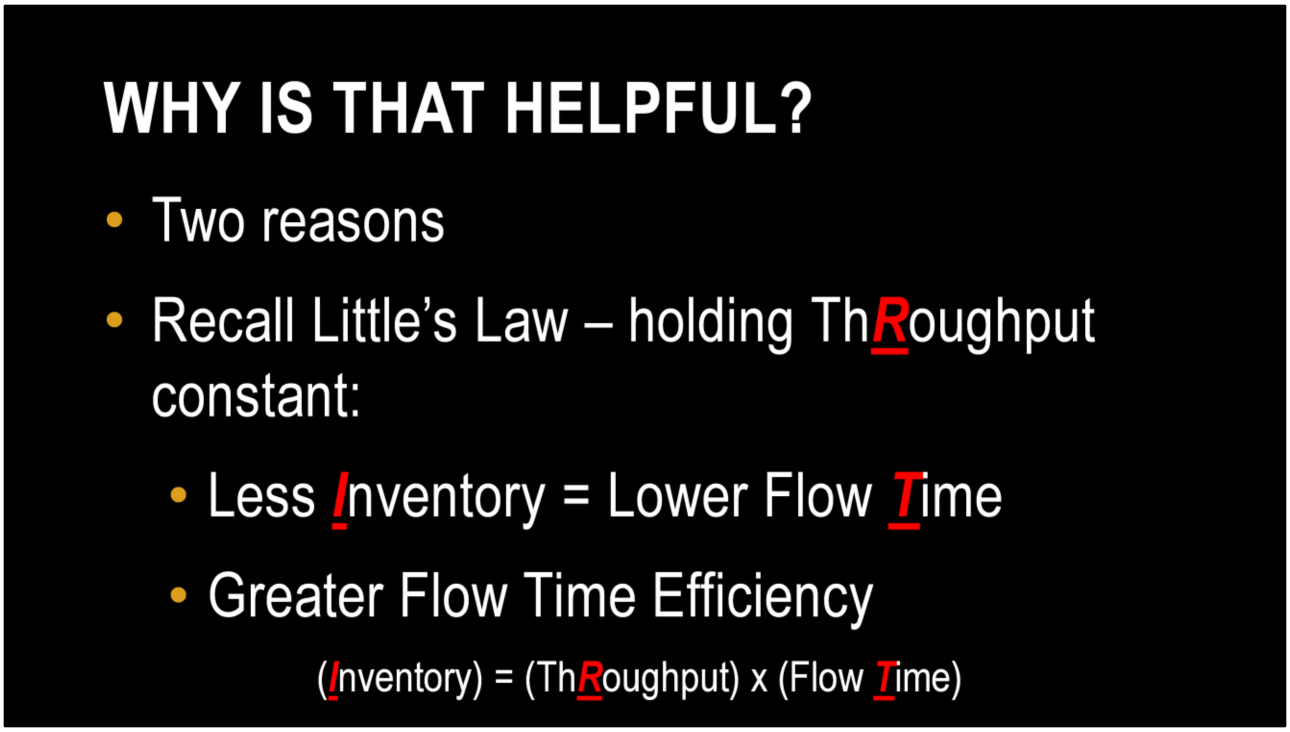
KANBAN (看板)

- Team members pull additional work when they are ready for it
- Work isn't forced on them
- The amount of inventory in the system is entirely controlled by how many cards are in circulation

Unlike a push-based flow, where assembled doors keep arriving at my station and I have to keep up, a kanban system means that I get work when I signal that I'm ready for it. This is why it's also known as "just in time" production.

KANBAN (看板)

- Over time, managers remove cards to maintain the absolute minimum level of inventory in circulation

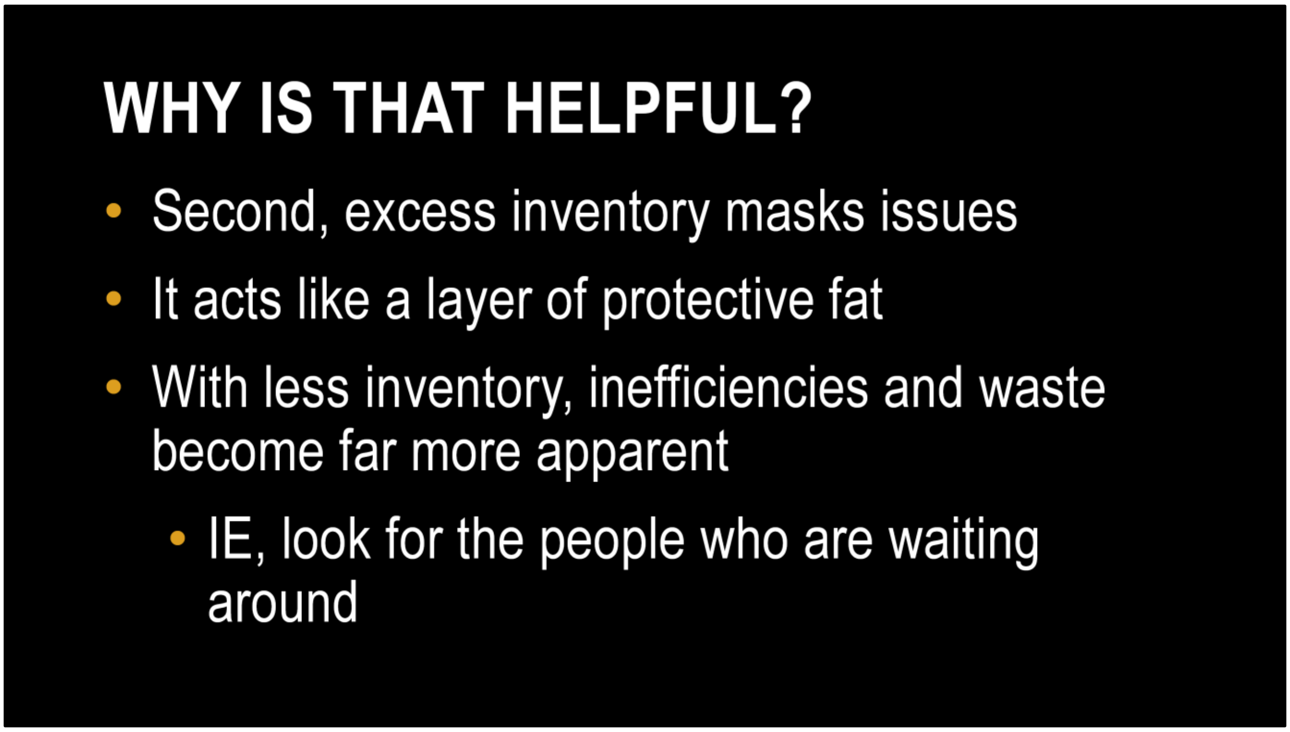


WHY IS THAT HELPFUL?

- Two reasons
- Recall Little's Law – holding ThRoughput constant:
  - Less Inventory = Lower Flow Time
  - Greater Flow Time Efficiency

(Inventory) = (ThRoughput) x (Flow Time)

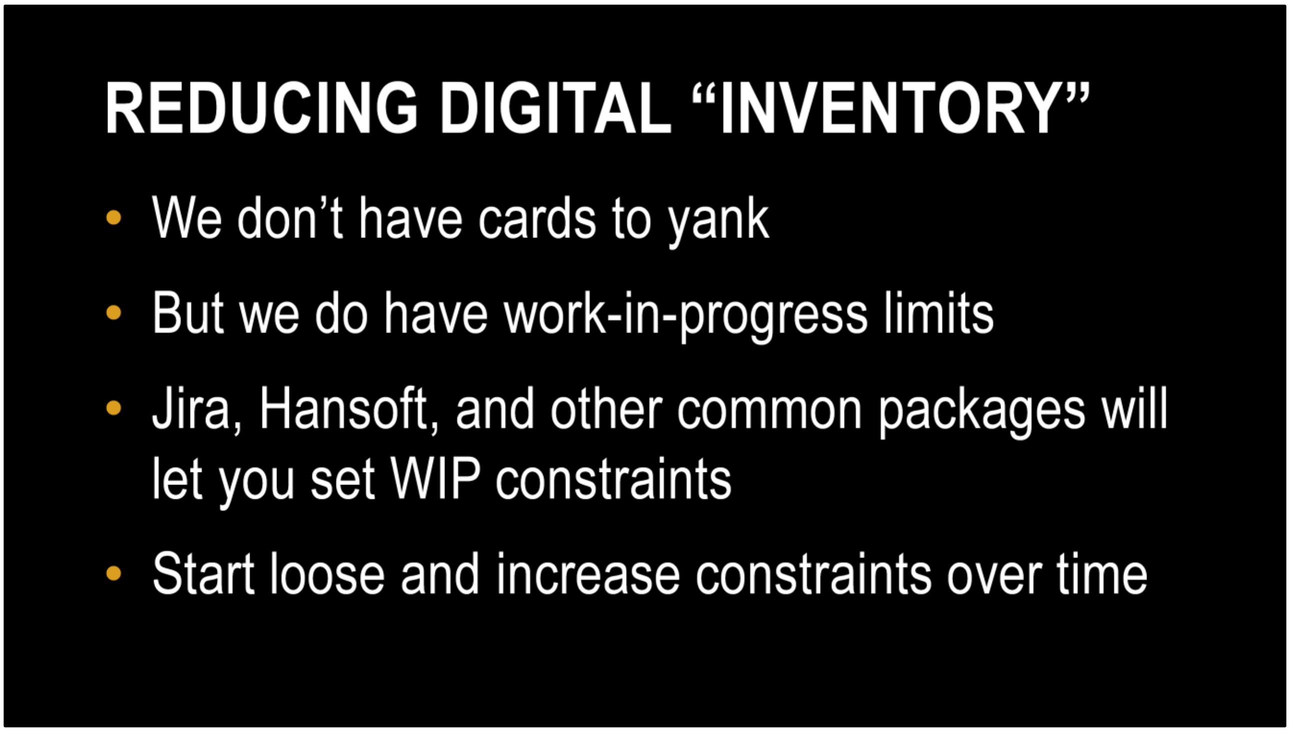
Assuming Toyota's throughput did not slow down, having less inventory means that any individual unit spends less time in the process because it spends less time waiting in queues. IE, your actual flow time moves closer to your theoretical flow time.



WHY IS THAT HELPFUL?

- Second, excess inventory masks issues
- It acts like a layer of protective fat
- With less inventory, inefficiencies and waste become far more apparent
  - IE, look for the people who are waiting around

If you have tons excess inventory lying around, there is always something to work on, so imbalances become harder to spot.



REDUCING DIGITAL "INVENTORY"

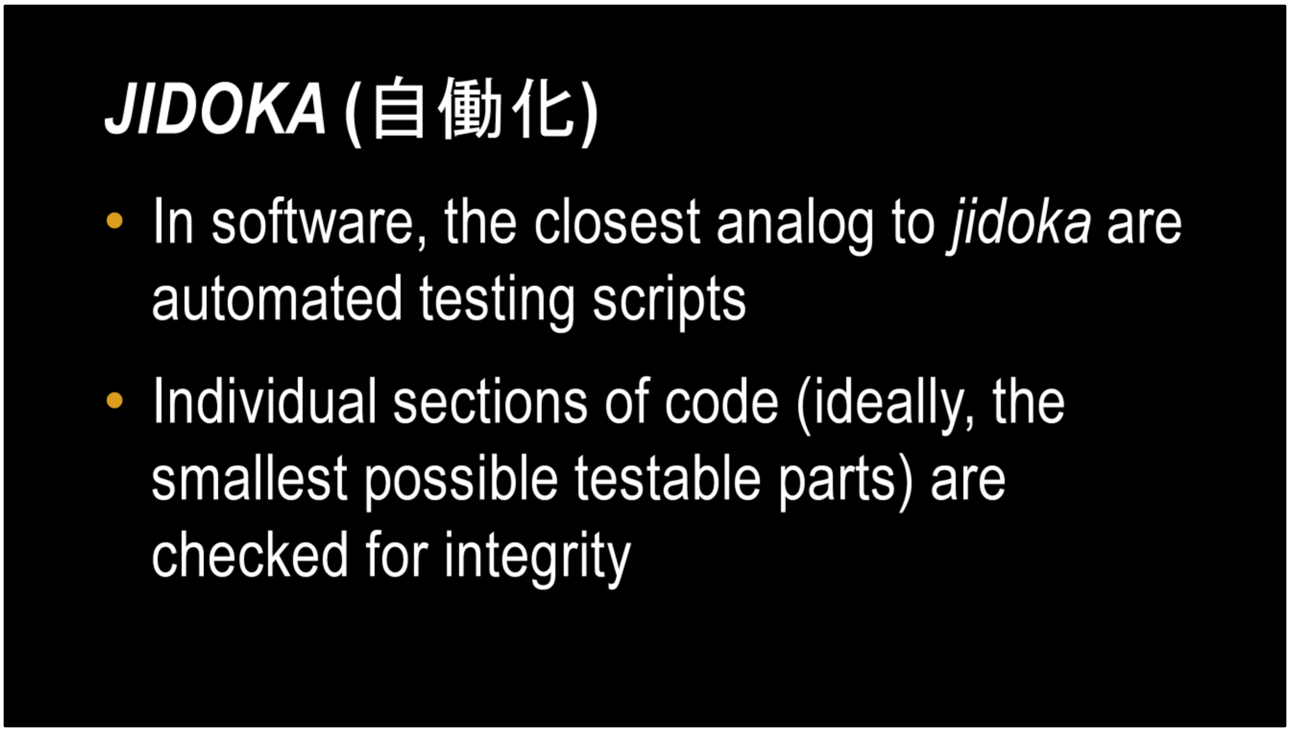
- We don't have cards to yank
- But we do have work-in-progress limits
- Jira, Hansoft, and other common packages will let you set WIP constraints
- Start loose and increase constraints over time

Use your software to limit the amount of tasks that can be assigned to any one person to simulate the effect of removing cards from the process.

JIDOKA
自働化

JIDOKA (自働化)

- “Autonomation” (literally, “automation with a human touch”)
- Machines and sensors designed to automatically detect and flag quality issues as they occur
- And stop production if necessary



JIDOKA (自働化)

- In software, the closest analog to *jidoka* are automated testing scripts
- Individual sections of code (ideally, the smallest possible testable parts) are checked for integrity

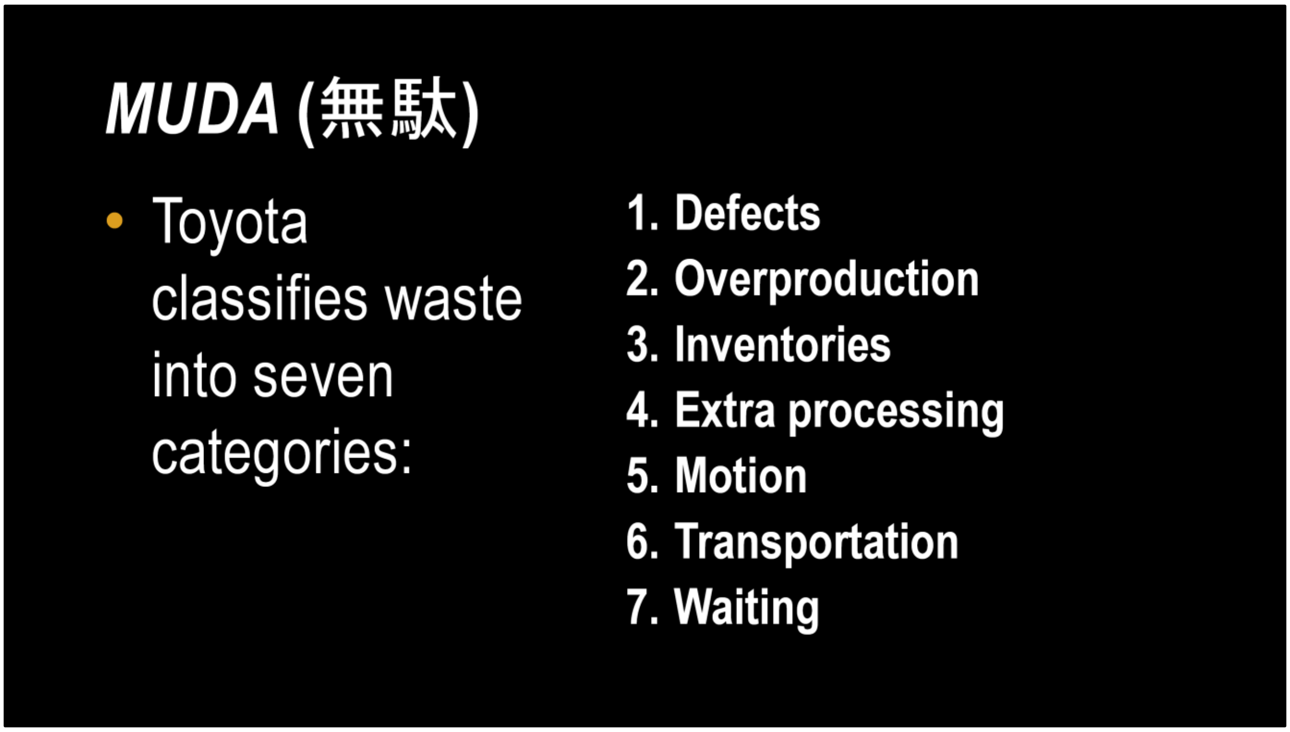
JIDOKA (自働化)

- Other examples:
  - Bots that play the game automatically
  - Robust crash reporting systems
  - Continuous integration

MUDA
無駄

MUDA (無駄)

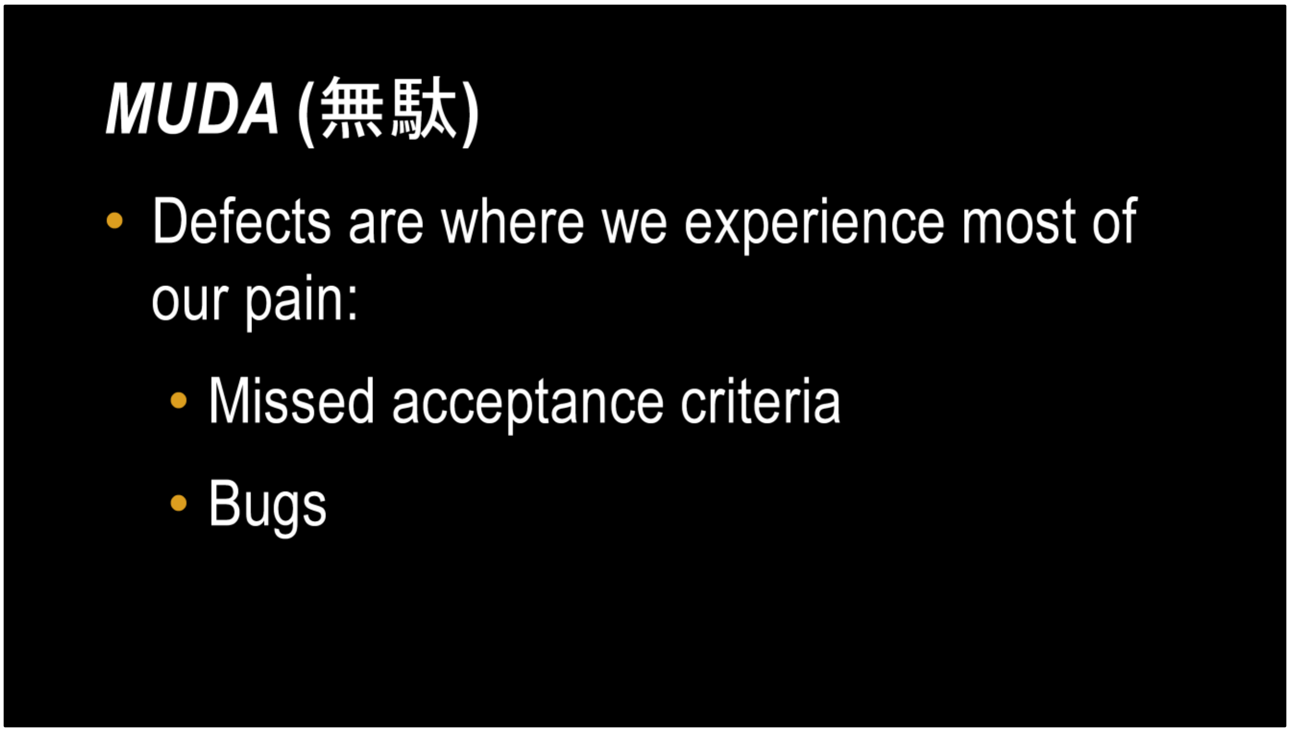
- Literally “waste”



MUDA (無駄)

- Toyota classifies waste into seven categories:

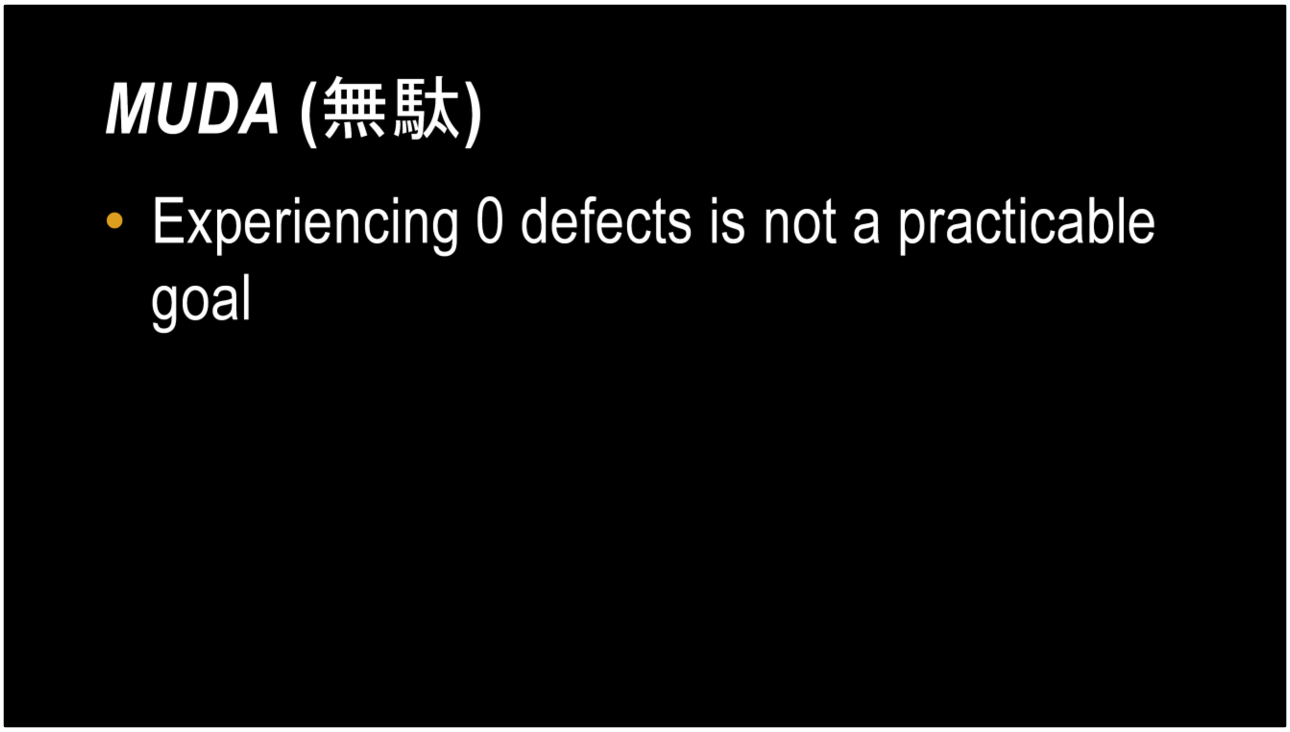
1. Defects
2. Overproduction
3. Inventories
4. Extra processing
5. Motion
6. Transportation
7. Waiting



MUDA (無駄)

- Defects are where we experience most of our pain:
  - Missed acceptance criteria
  - Bugs

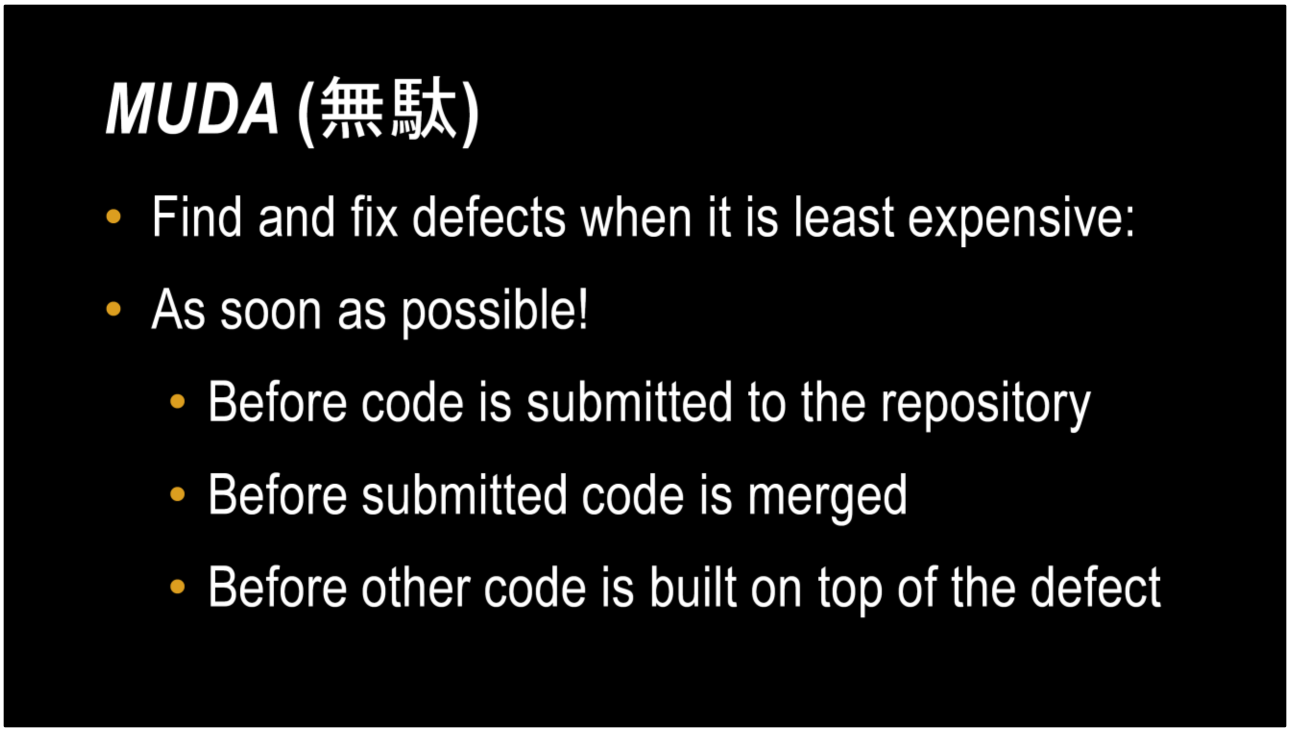
Meetings are a close second in my experience, but we’ll stick with defects for this presentation



MUDA (無駄)

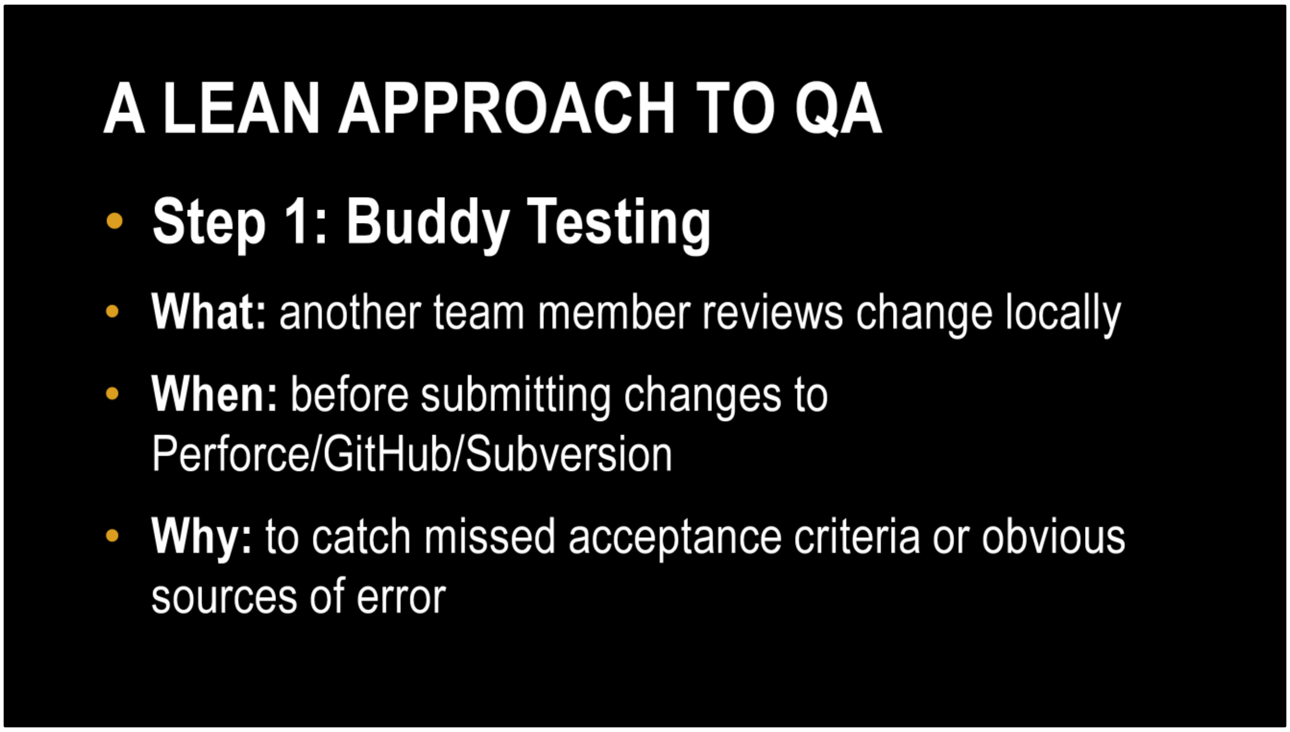
- Experiencing 0 defects is not a practicable goal

People will make mistakes



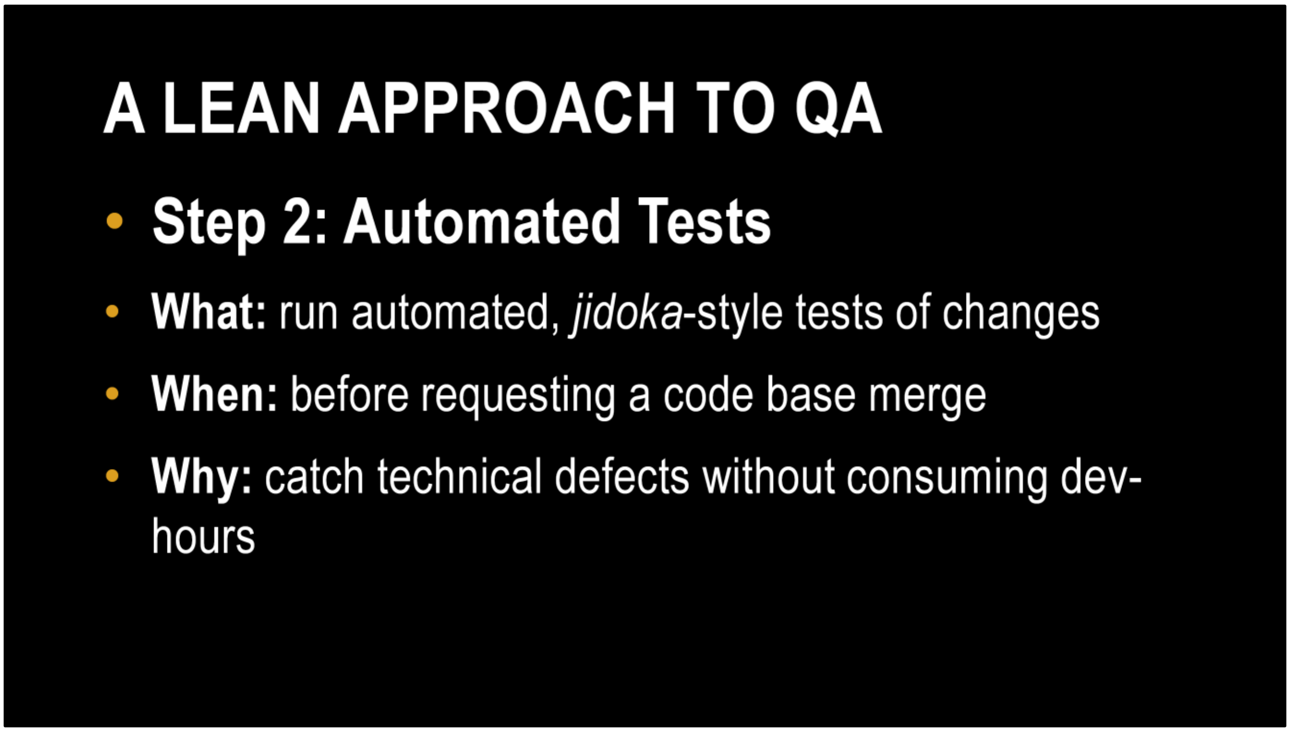
MUDA (無駄)

- Find and fix defects when it is least expensive:
- As soon as possible!
  - Before code is submitted to the repository
  - Before submitted code is merged
  - Before other code is built on top of the defect



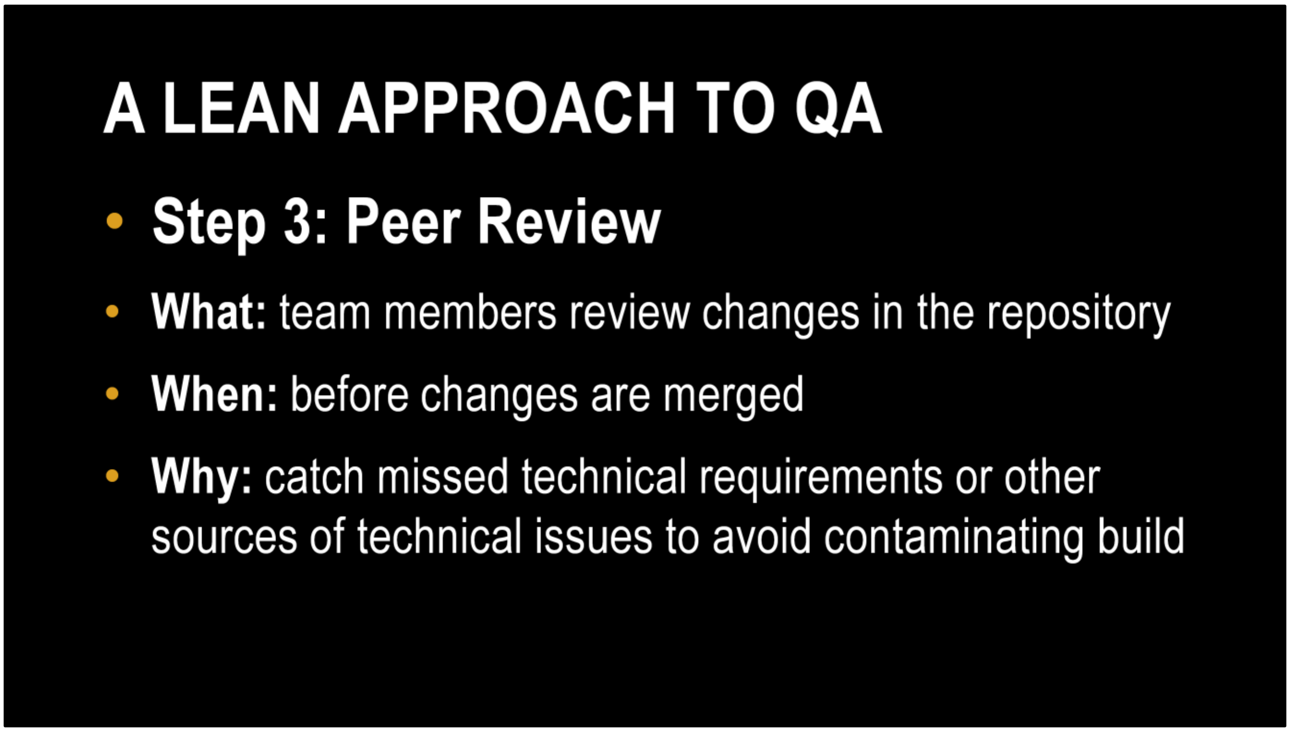
A LEAN APPROACH TO QA

- **Step 1: Buddy Testing**
- **What:** another team member reviews change locally
- **When:** before submitting changes to Perforce/GitHub/Subversion
- **Why:** to catch missed acceptance criteria or obvious sources of error



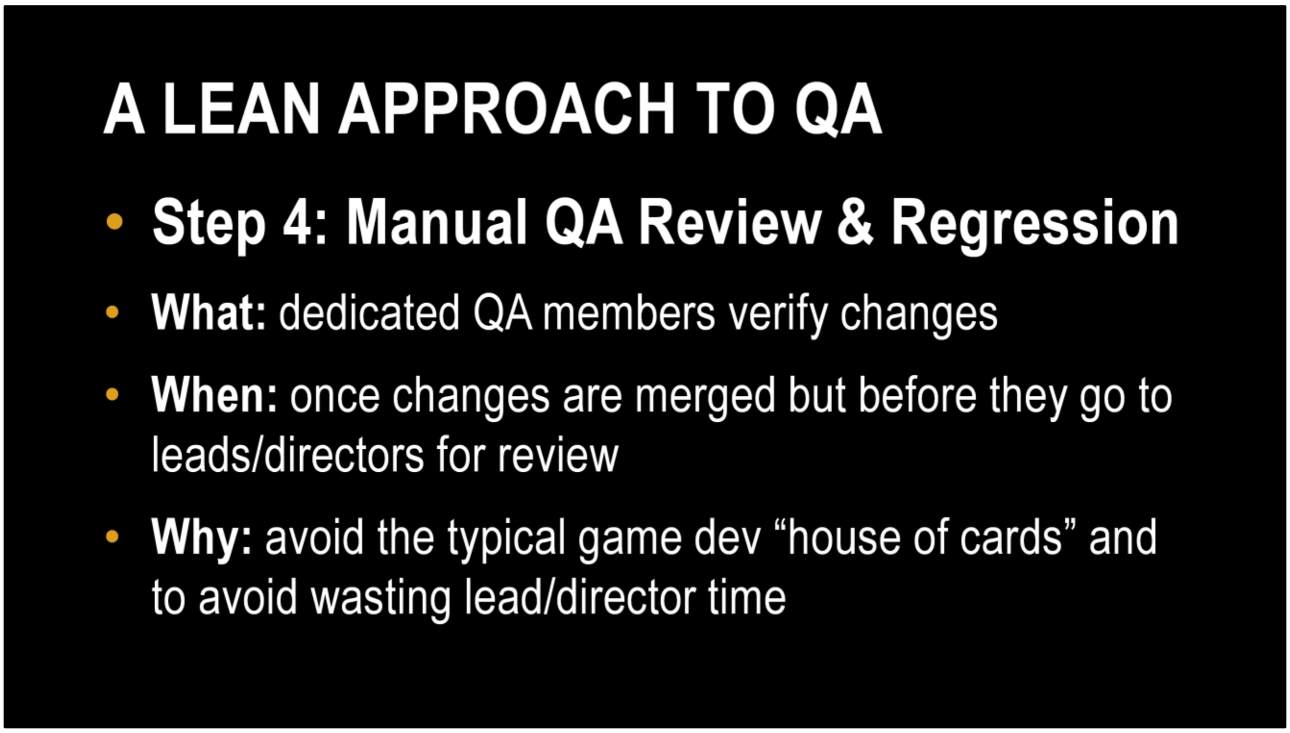
A LEAN APPROACH TO QA

- **Step 2: Automated Tests**
- **What:** run automated, *jidoka*-style tests of changes
- **When:** before requesting a code base merge
- **Why:** catch technical defects without consuming dev-hours



A LEAN APPROACH TO QA

- **Step 3: Peer Review**
- **What:** team members review changes in the repository
- **When:** before changes are merged
- **Why:** catch missed technical requirements or other sources of technical issues to avoid contaminating build

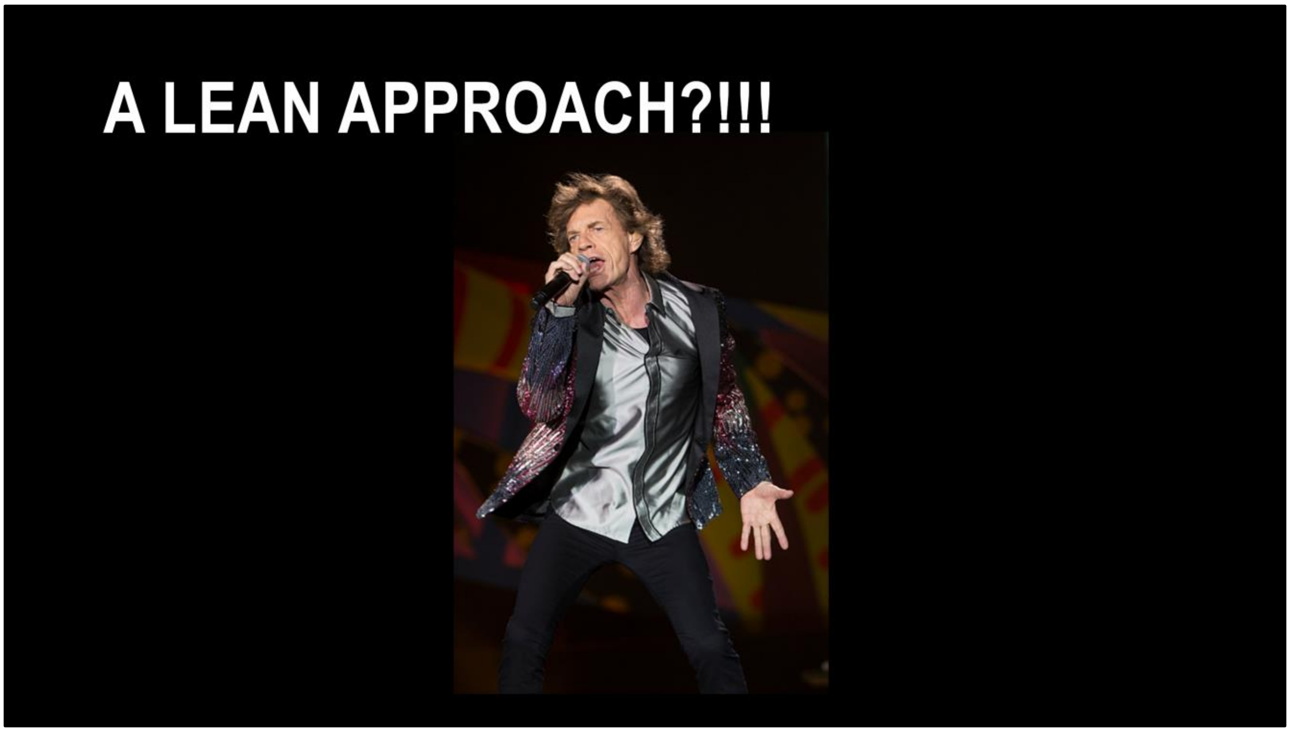


A LEAN APPROACH TO QA

- **Step 4: Manual QA Review & Regression**
- **What:** dedicated QA members verify changes
- **When:** once changes are merged but before they go to leads/directors for review
- **Why:** avoid the typical game dev “house of cards” and to avoid wasting lead/director time

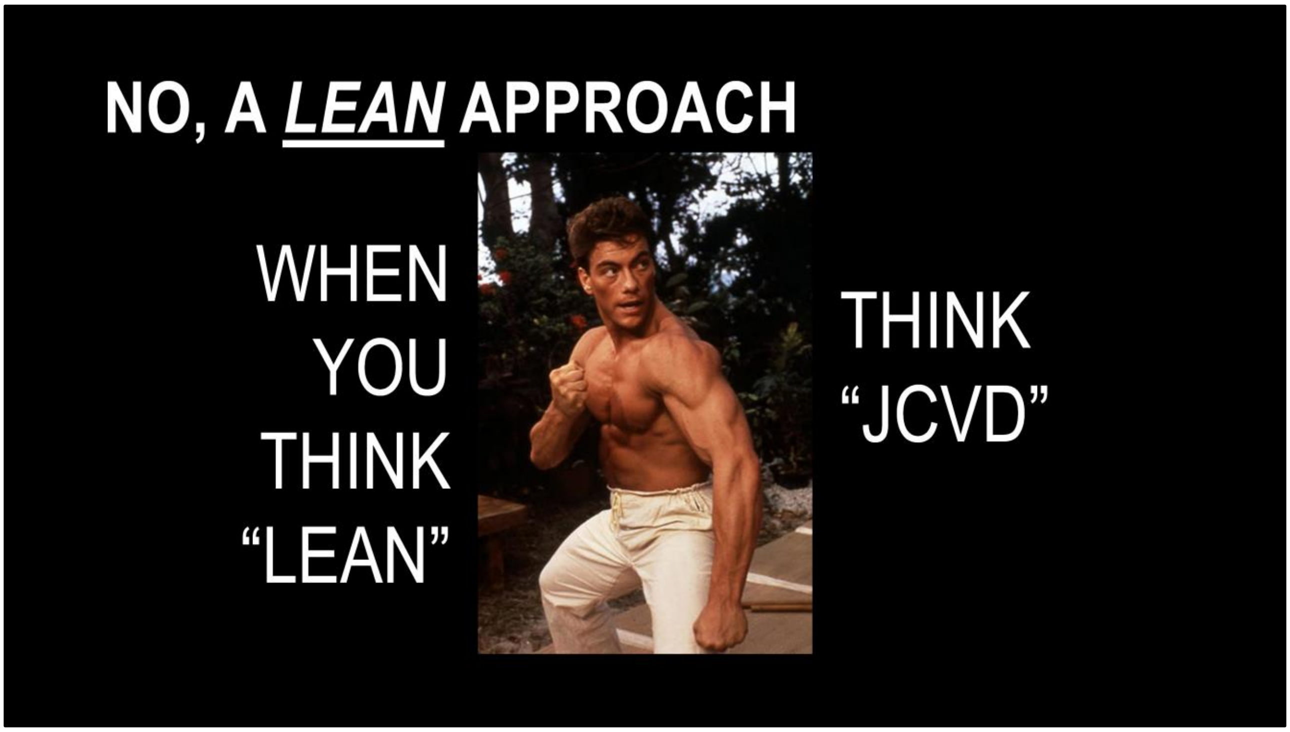
A LEAN APPROACH?!!!

1) The previous 4 slides might sound onerous, but I have seen this disciplined approach to QA in action, and it results in an amazingly clean and stable code database
2) You might have trouble reconciling a 4-step process with something called “lean production”, because when you think “lean” you might be thinking of…



A LEAN APPROACH?!!!

…this guy. But when I hear “lean” I think of…



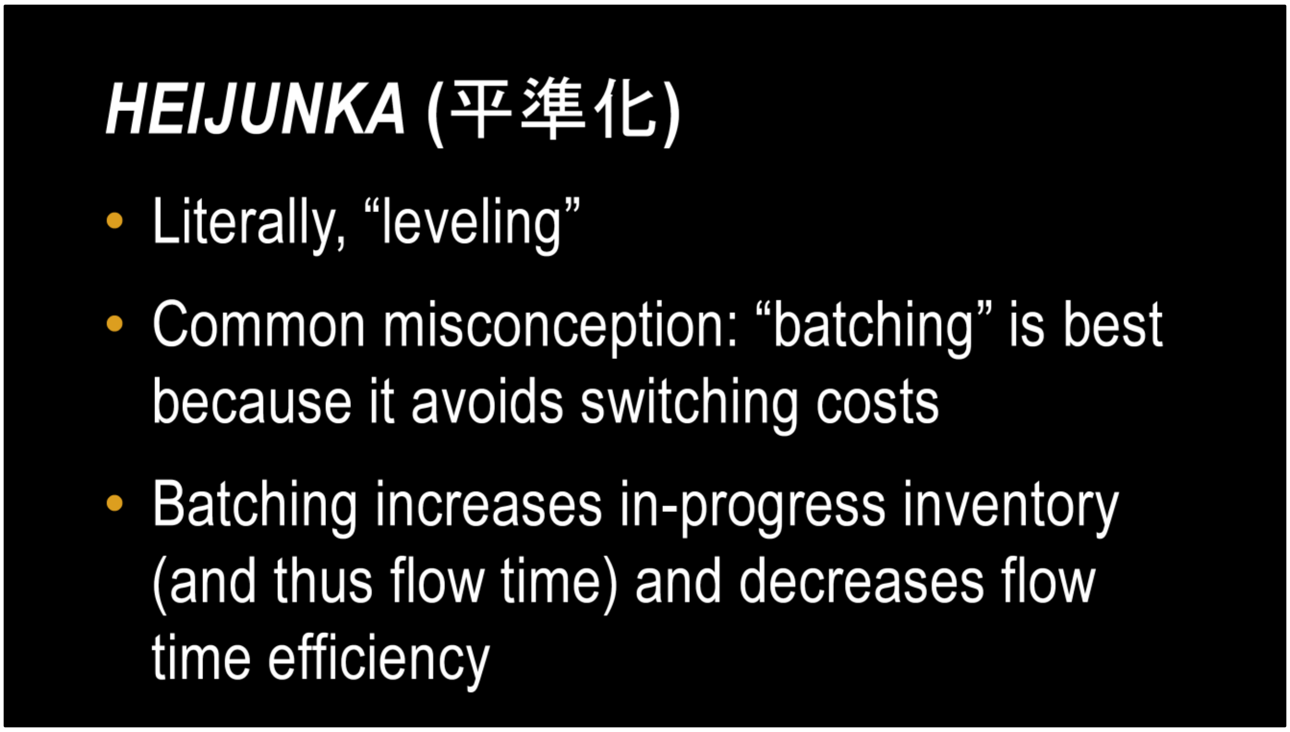
NO, A *LEAN* APPROACH

WHEN YOU THINK "LEAN"

THINK "JCVD"

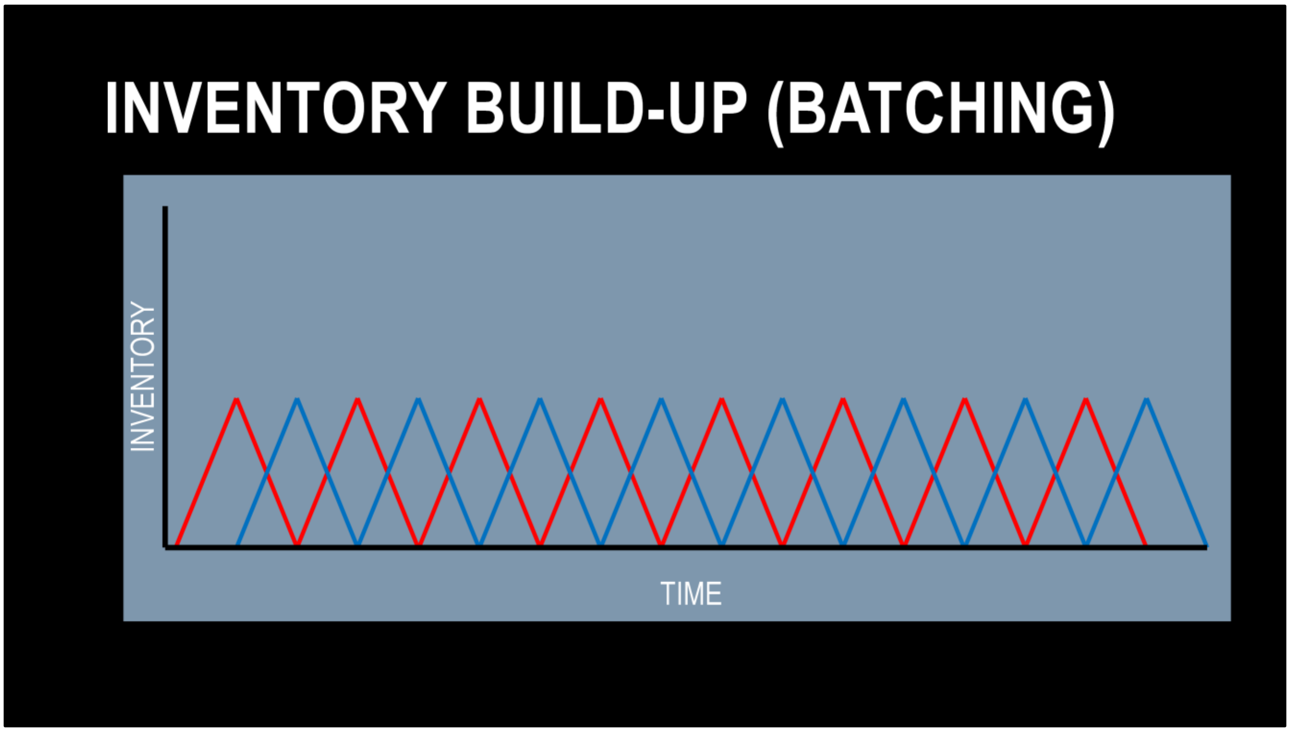
…this guy: muscular, technically proficient and disciplined, and oh-so-handsome.

HEIJUNKA
平準化



HEIJUNKA (平準化)

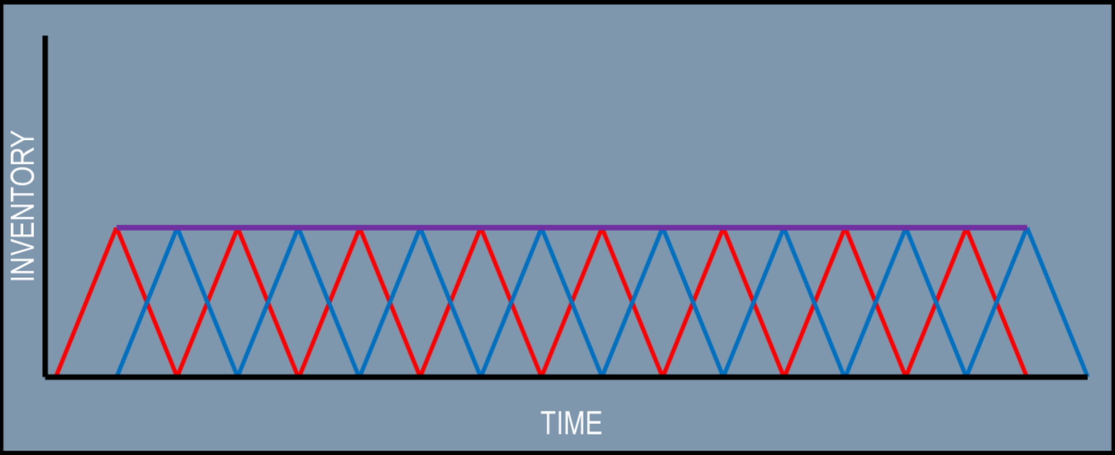
- Literally, “leveling”
- Common misconception: “batching” is best because it avoids switching costs
- Batching increases in-progress inventory (and thus flow time) and decreases flow time efficiency



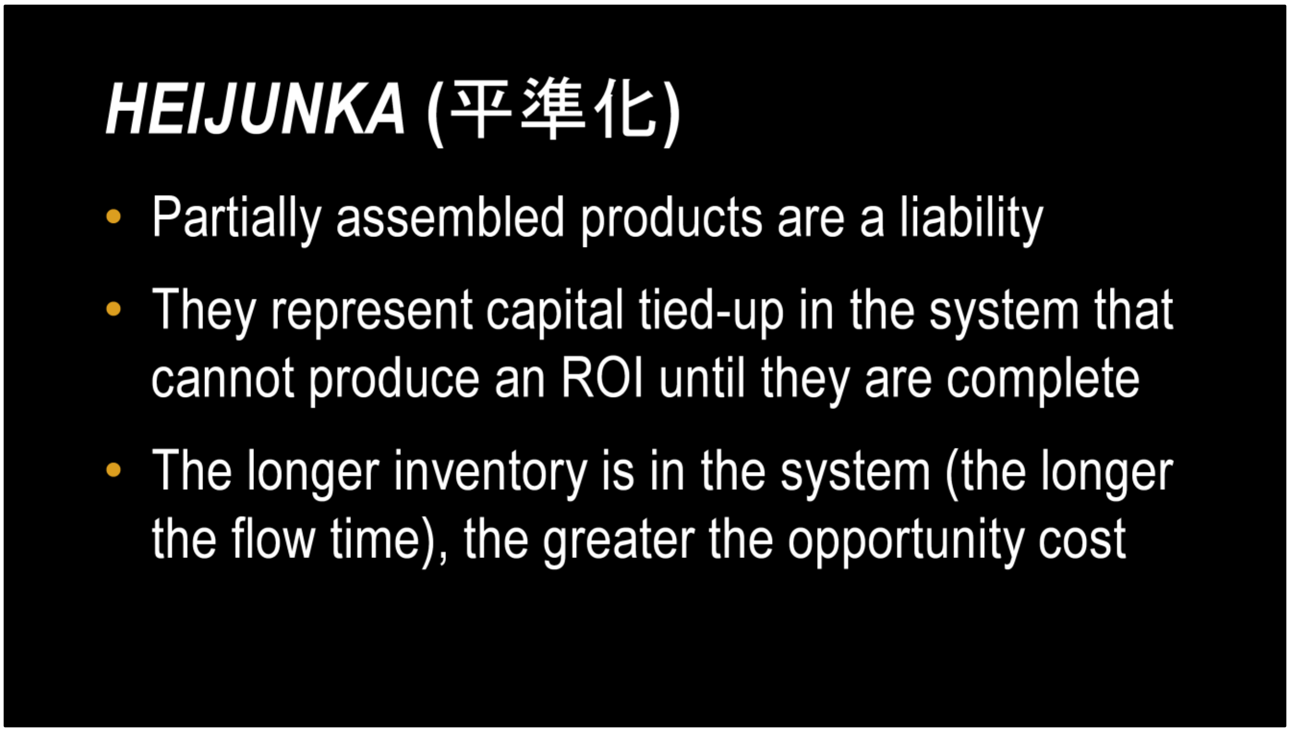
INVENTORY BUILD-UP (BATCHING)

As you prepare the Red Batch, you build up a bunch of in-progress inventory to avoid incurring a switching cost. Then, as you bulk process the Red Batch, you simultaneously start accruing inventory for the Blue Batch. Repeat ad infinitum.

INVENTORY BUILD-UP (BATCHING)

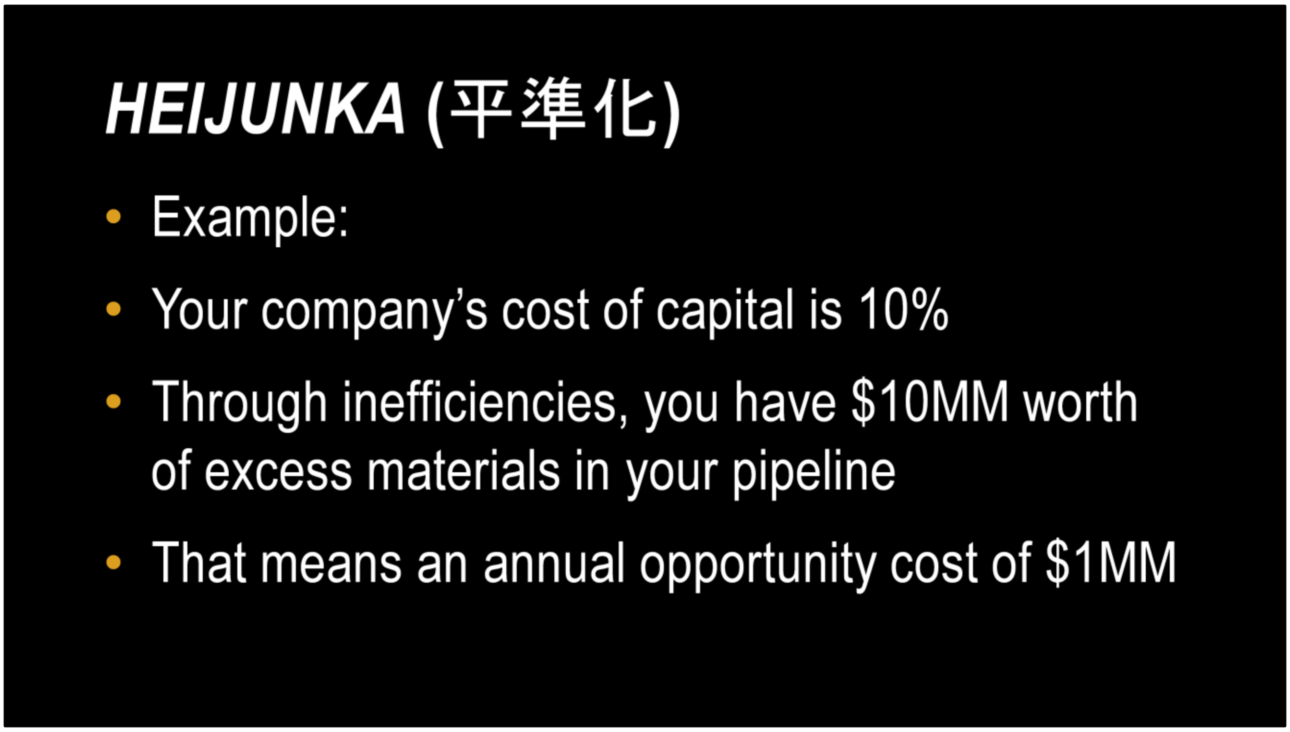


This leads to a perpetually inventory level (the purple line)



HEIJUNKA (平準化)

- Partially assembled products are a liability
- They represent capital tied-up in the system that cannot produce an ROI until they are complete
- The longer inventory is in the system (the longer the flow time), the greater the opportunity cost



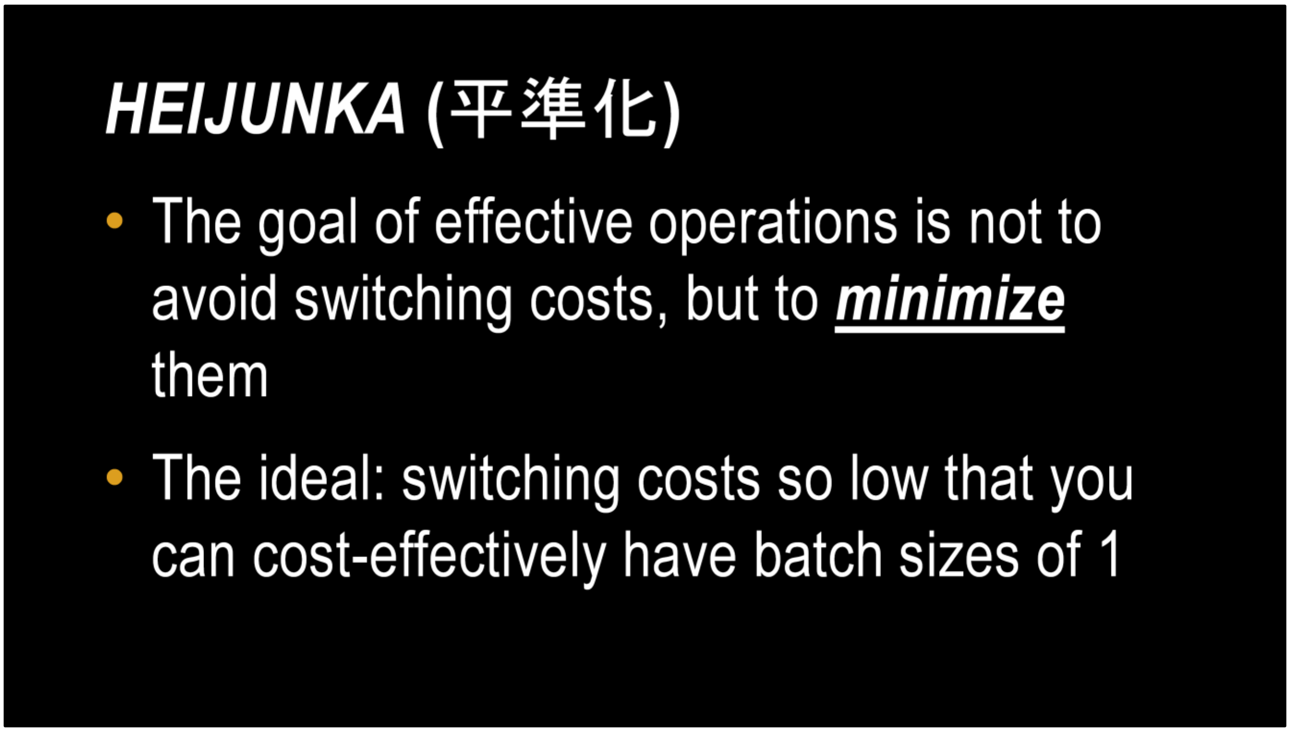
HEIJUNKA (平準化)

- Example:
- Your company's cost of capital is 10%
- Through inefficiencies, you have $10MM worth of excess materials in your pipeline
- That means an annual opportunity cost of $1MM

For the folks who aren't finance nerds: "cost of capital" mean, in simple terms, the return that the company expects to receive from any investment. So, for every $10 it invests, it expects to make a profit of $1 per year. In this case, the company could reasonably expect to make $1MM profit yearly on the $10MM of bloat currently in the system if you could somehow liberate it.

HEIJUNKA (平準化)

- Additionally, smaller, faster batches make it easier to catch recurring defects before they have a chance to propagate
- EG: a recurring defect results in much less waste if it impacts a batch of 5 units, versus a batch of 100

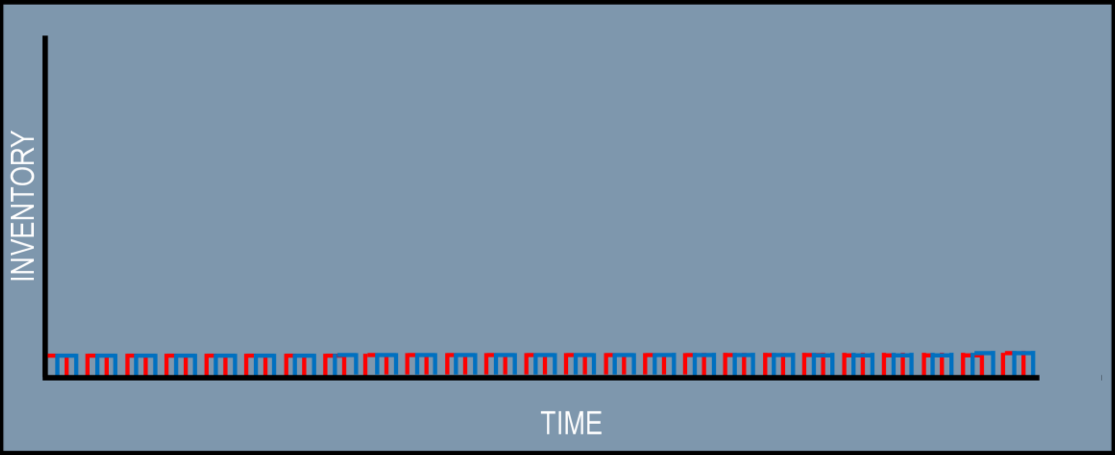


HEIJUNKA (平準化)

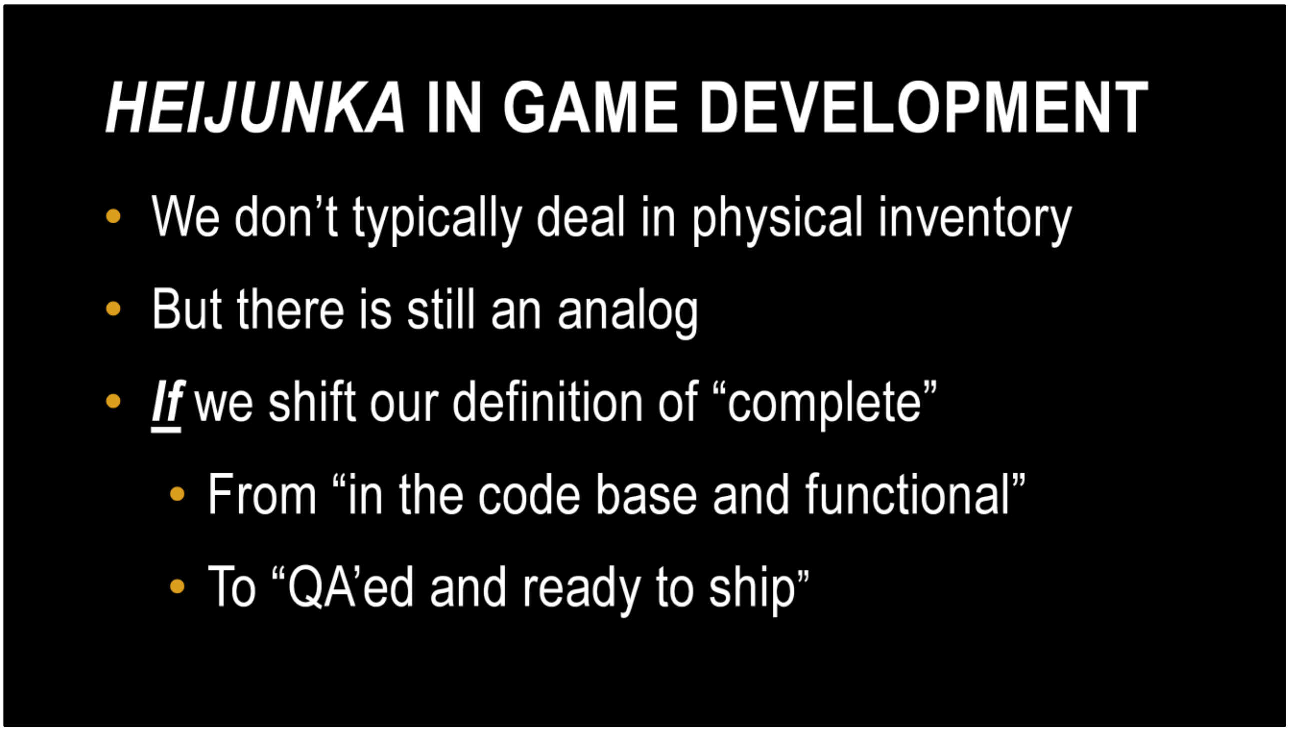
- The goal of effective operations is not to avoid switching costs, but to ***minimize*** them
- The ideal: switching costs so low that you can cost-effectively have batch sizes of 1

Toyota got down to batch sizes of one car

INVENTORY BUILD-UP (IDEAL)



In an efficient system, minimum inventory levels are collected and they are processed as quickly as possible



HEIJUNKA IN GAME DEVELOPMENT

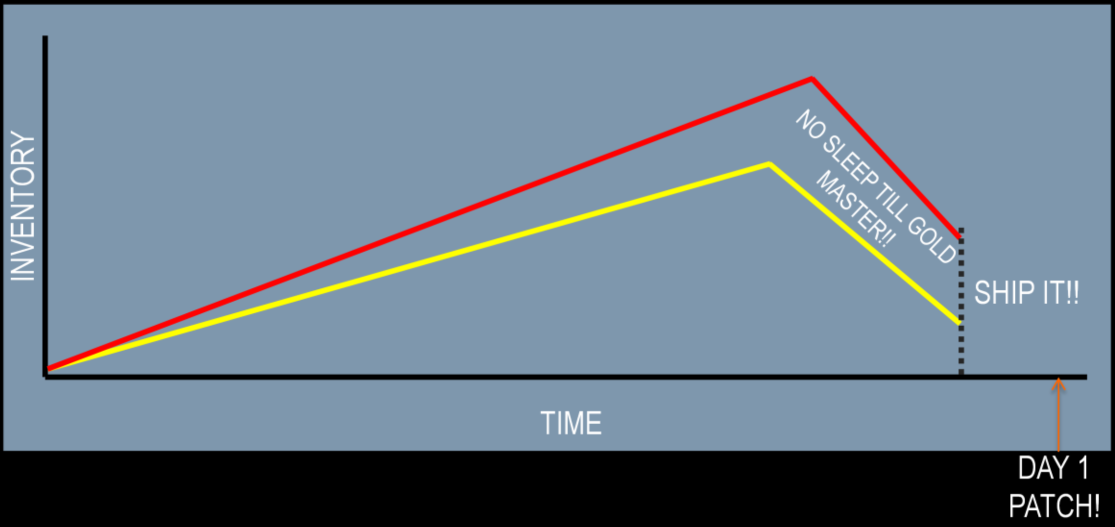
- We don't typically deal in physical inventory
- But there is still an analog
- ***If*** we shift our definition of "complete"
  - From "in the code base and functional"
  - To "QA'ed and ready to ship"

The analog becomes more apparent if we include a QA pass in our definition of our feature being complete

HEIJUNKA IN GAME DEVELOPMENT

- Every incomplete feature is a liability
- It can’t generate value for gamers until it is (at least largely) defect free
- It represents an investment of man-hours that can’t produce a return on investment until you run it through QA

GAME DEV BUILD-UP

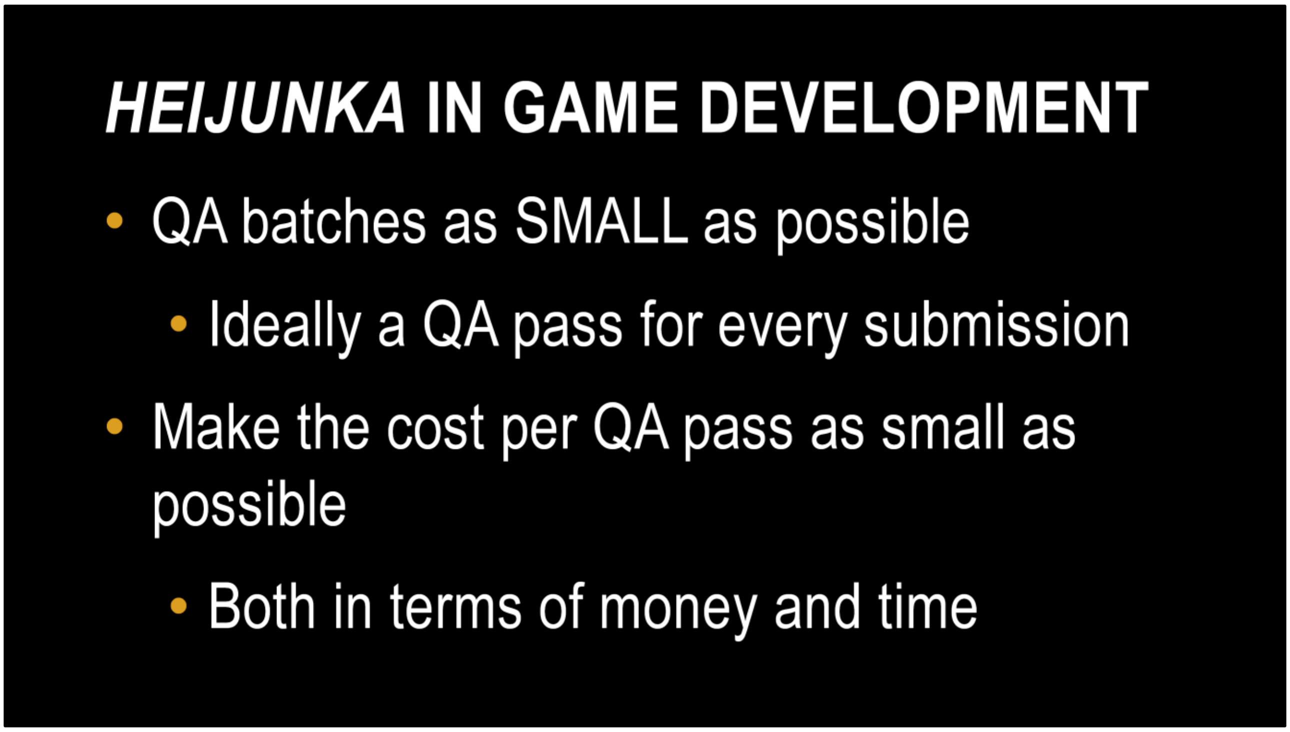


- In a typical process, we generate an inventory of incomplete (not QA'ed) features (yellow line)
- And simultaneously we accrue defects at some multiple of those features
- Then we hit our alpha/beta phases and focus on squashing bugs, and nobody sleeps, and we witness the house of cards because every fix breaks five other things
- And finally we throw our hands up and say "SHIP IT!!" and resign ourselves to a day-1 patch

In short, we use an insane amount of batching: we batch process an entire game's worth of QA

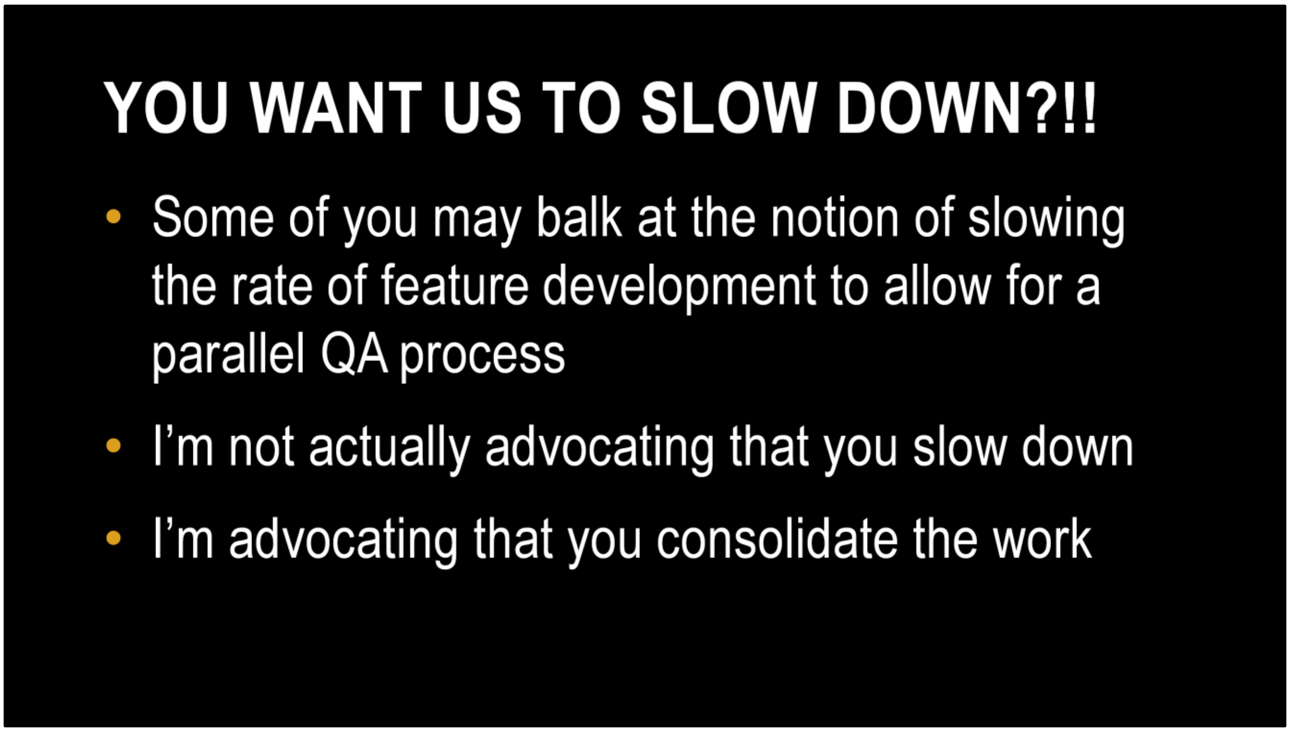
HEIJUNKA IN GAME DEVELOPMENT

- Ensure features are defect-free when they are added to the code base
- Move from a “build->build->build->build->build->FIX!!!!!” mentality
  - IE, production->alpha->beta->certification
- Into a “build->fix->build->fix->build->fix” cadence



HEIJUNKA IN GAME DEVELOPMENT

- QA batches as SMALL as possible
  - Ideally a QA pass for every submission
- Make the cost per QA pass as small as possible
  - Both in terms of money and time

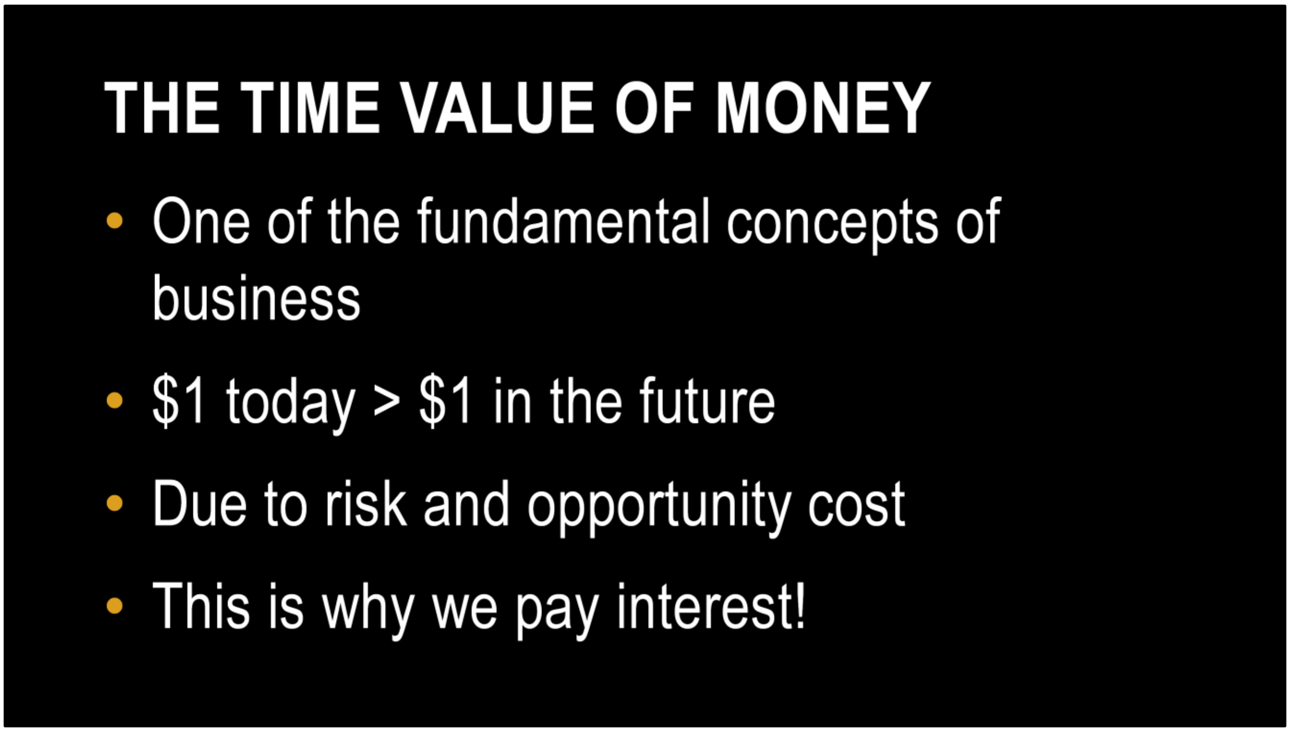


YOU WANT US TO SLOW DOWN?!!

- Some of you may balk at the notion of slowing the rate of feature development to allow for a parallel QA process
- I'm not actually advocating that you slow down
- I'm advocating that you consolidate the work

The analogy I use is defragging a hard-drive: we should de-frag our production process to make QA a part of feature development, rather than a separate phase of production.

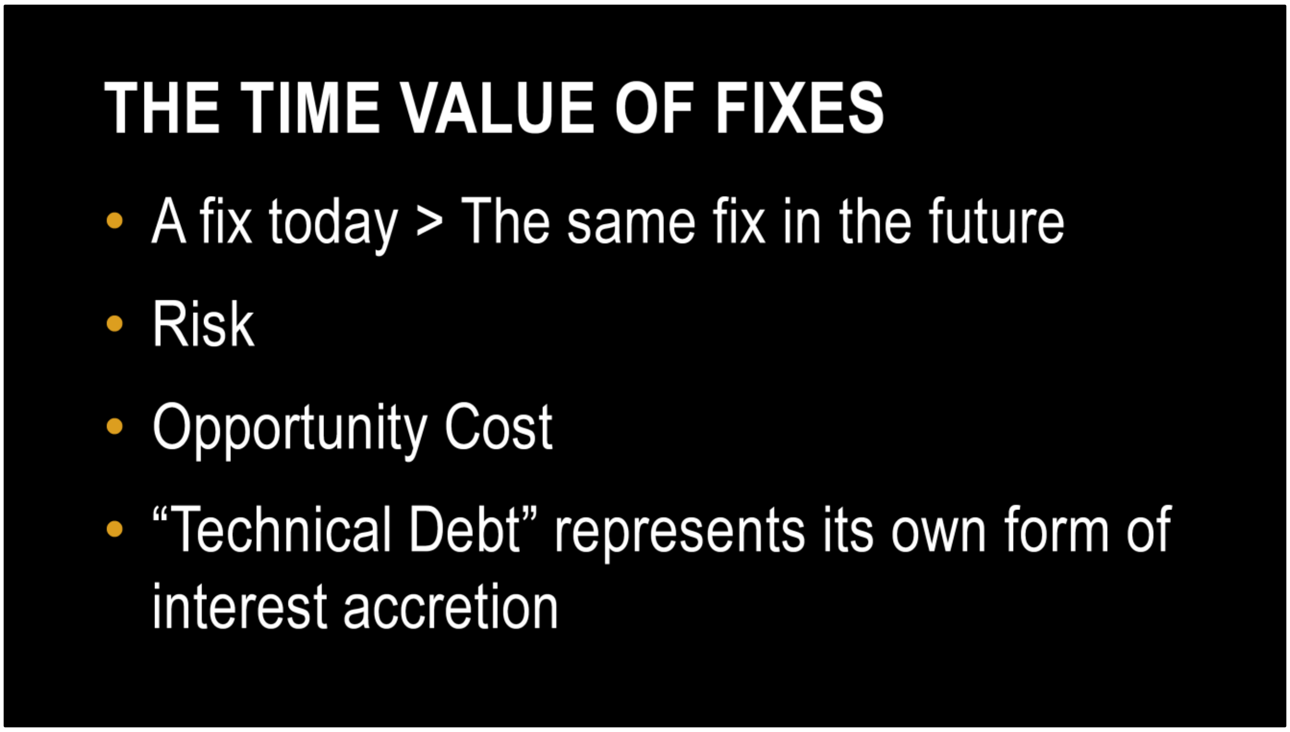
And here's another analogy I like to employ on this subject…


THE TIME VALUE OF MONEY

- One of the fundamental concepts of business
- $1 today > $1 in the future
- Due to risk and opportunity cost
- This is why we pay interest!

Example: I can get a $1 today or $1 in six months. The promise of a dollar in six months is less valuable because I don't know if I will actually get the dollar (risk) and there will be things I could have done with the dollar in the meantime (opportunity cost).

The primary reason you pay interest is to compensate your bank or credit card company for the risk and opportunity cost of providing you with a mortgage or a line of credit.



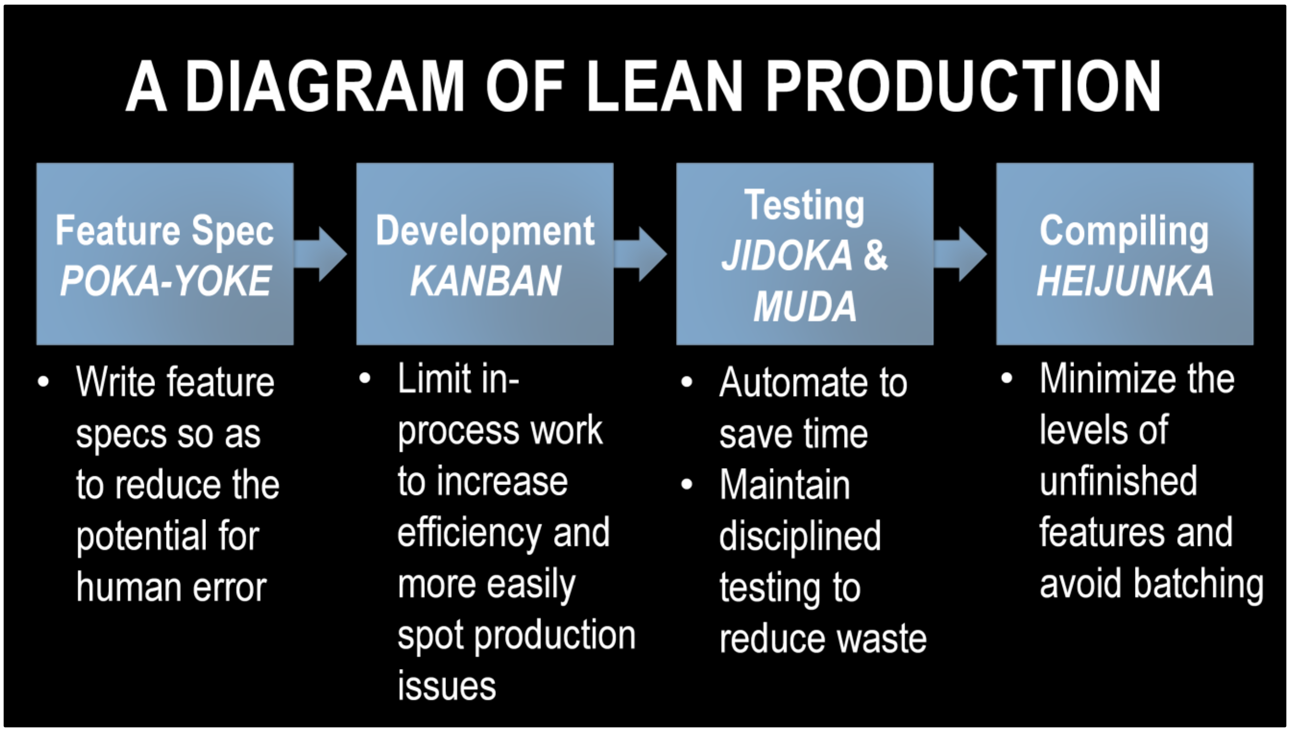
THE TIME VALUE OF FIXES

- A fix today > The same fix in the future
- Risk
- Opportunity Cost
- “Technical Debt” represents its own form of interest accretion

In software development, I like to think in terms of something I call “the time value of fixes”. A fix today is more valuable than that same fix in the future, for the same reasons:

- Risk – the fix might be larger in the future, because more code is built around it
- Opportunity cost – if the fix is larger in the future, it will consumer more time that could have gone to other efforts

I’m not alone in seeing analogs between finance and development. The term “technical debt” represents a form of compounding interest. If you write hacky, short-cut code, you are running up a charge on your production credit card. The longer that debt stays on the books – the more code gets built around it – the more expensive it will be to zero it out: the necessary re-factor will be larger.



A DIAGRAM OF LEAN PRODUCTION

Feature Spec ***POKA-YOKE*** → **Development** ***KANBAN*** → **Testing** ***JIDOKA & MUDA*** → **Compiling** ***HEIJUNKA***

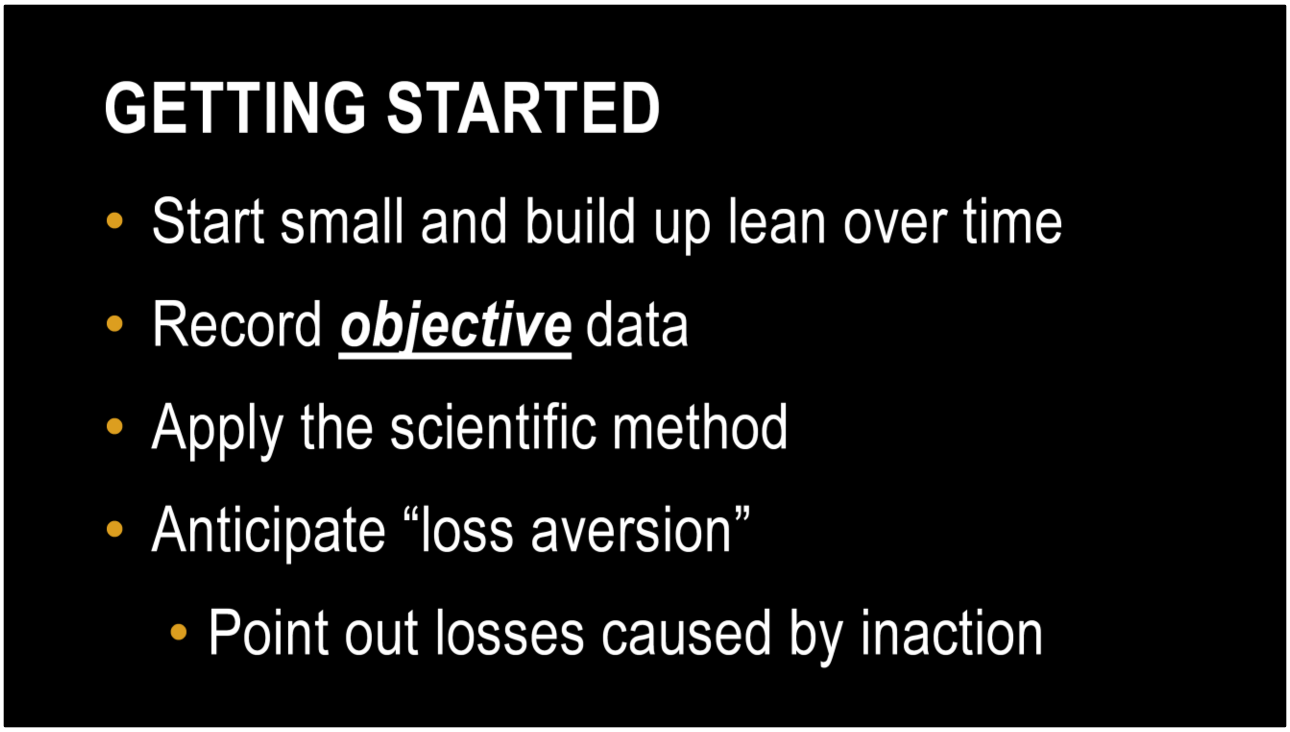
Feature Spec *POKA-YOKE*
- Write feature specs so as to reduce the potential for human error

Development *KANBAN*
- Limit in-process work to increase efficiency and more easily spot production issues

Testing *JIDOKA & MUDA*
- Automate to save time
- Maintain disciplined testing to reduce waste

Compiling *HEIJUNKA*
- Minimize the levels of unfinished features and avoid batching

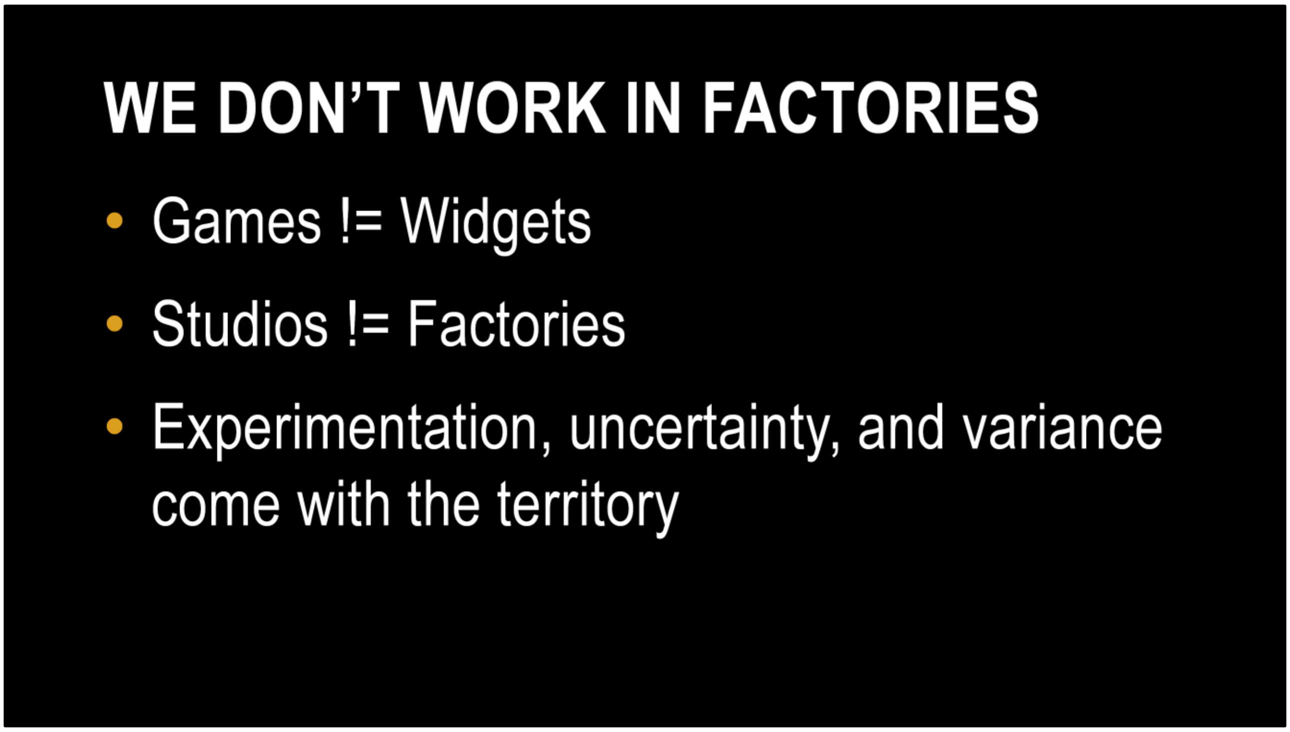
An overview of lean development, with the prior five terms in context.


GETTING STARTED

- Start small and build up lean over time
- Record ***objective*** data
- Apply the scientific method
- Anticipate “loss aversion”
  - Point out losses caused by inaction

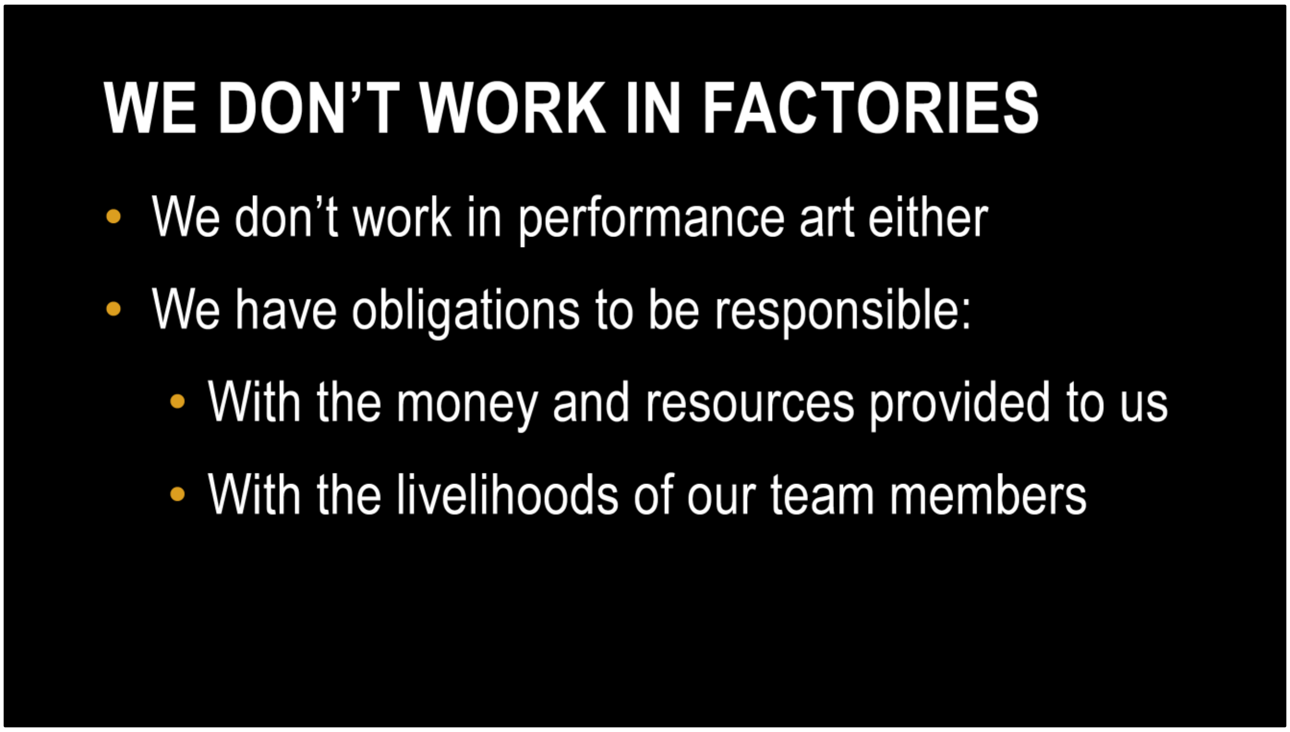
- Regarding objective data: it’s very tempting to just say “things are better”, because that’s easy. Be disciplined: record objective measurements and gauge the impact of an change.
- Regarding the scientific method: Question -> Hypothesis -> Test -> Observation -> Analysis -> Conclusion -> Question
  - Hand-in-hand with recording objective data, set objective hypotheses (EG, this change should increase throughput by at least 10%)
- In my experience as a manger, consultant, and scrum master, loss aversion is the largest hurdle to adopting a new modus operandi
  - Losses loom larger than gains in our minds
  - People will fixate on what a change costs instead of considering what they’ll gain
  - The trick with loss aversion is that it’s like putting out an oil well fire: you have to set off a bigger explosion next to it
    - IE, instead of communicating what they’ll gain from the change, communicate what they’ll lose if they don’t change

CLOSING THOUGHTS



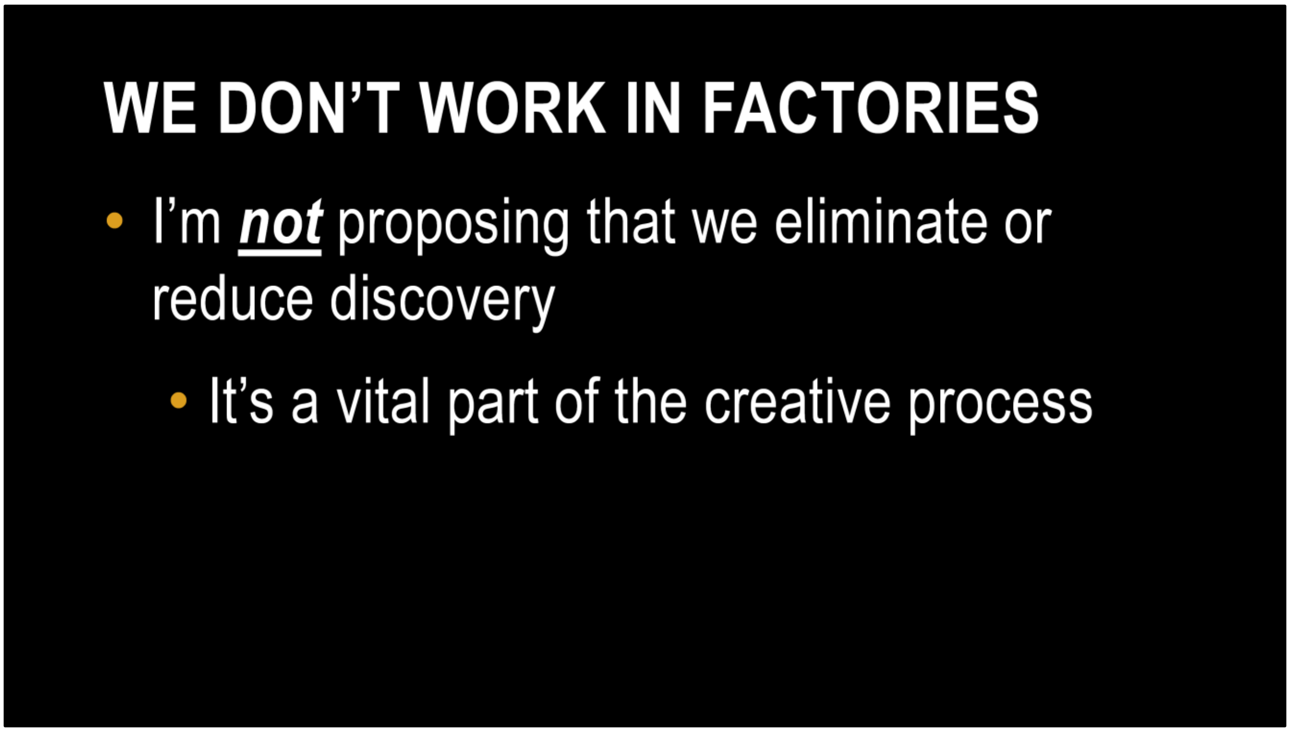
WE DON'T WORK IN FACTORIES

- Games != Widgets
- Studios != Factories
- Experimentation, uncertainty, and variance come with the territory



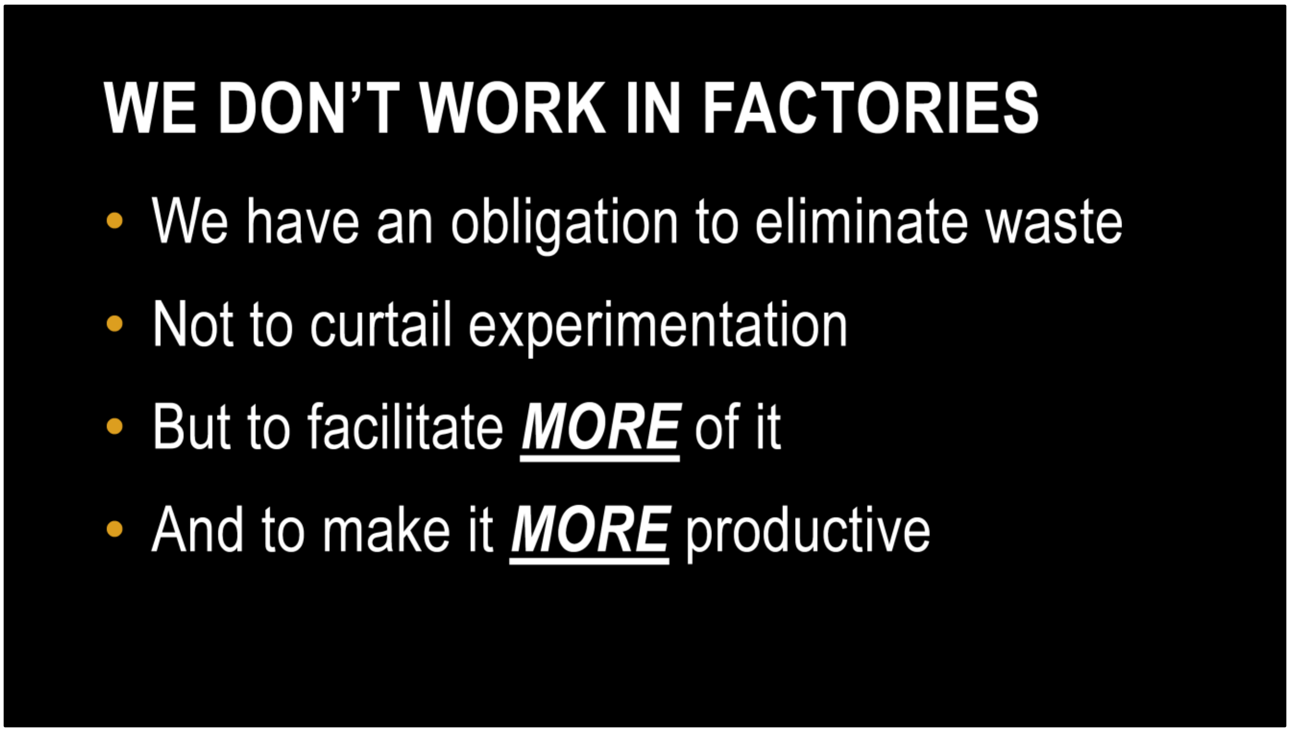
WE DON’T WORK IN FACTORIES

- We don’t work in performance art either
- We have obligations to be responsible:
  - With the money and resources provided to us
  - With the livelihoods of our team members



WE DON’T WORK IN FACTORIES

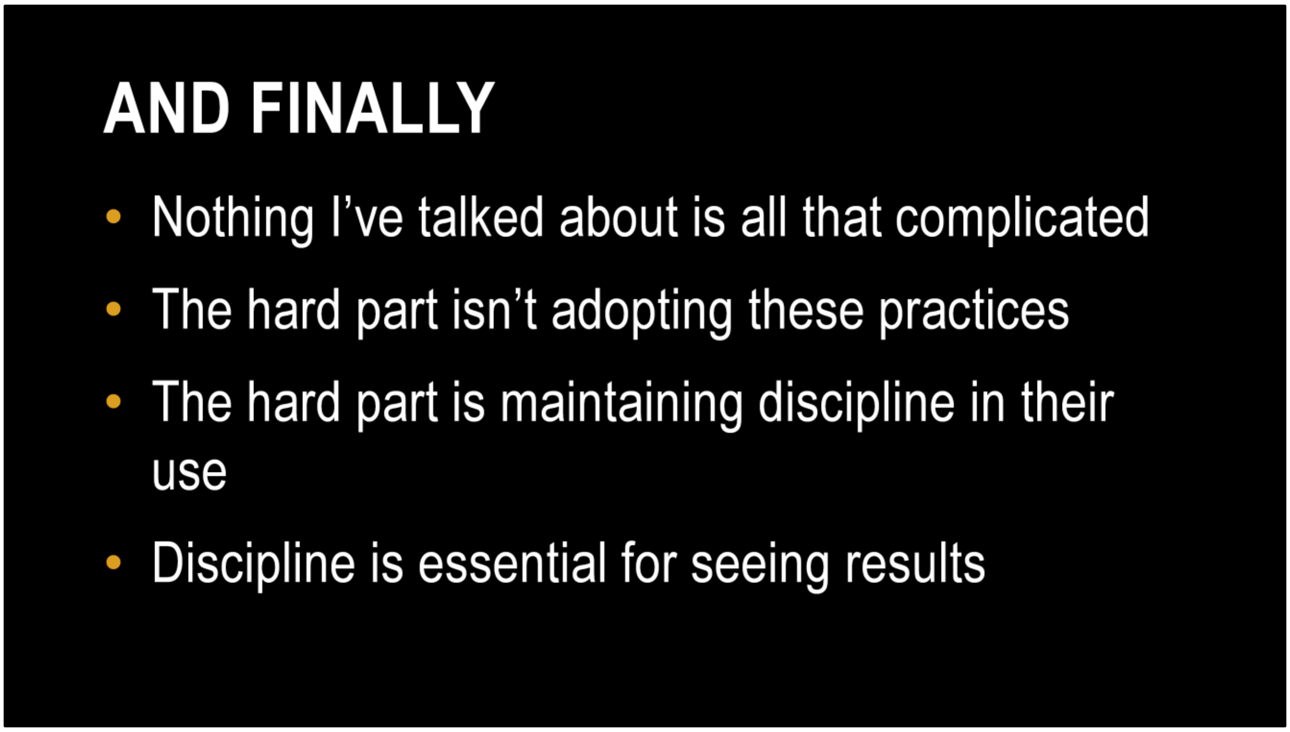
- I’m ***not*** proposing that we eliminate or reduce discovery
  - It’s a vital part of the creative process



WE DON’T WORK IN FACTORIES

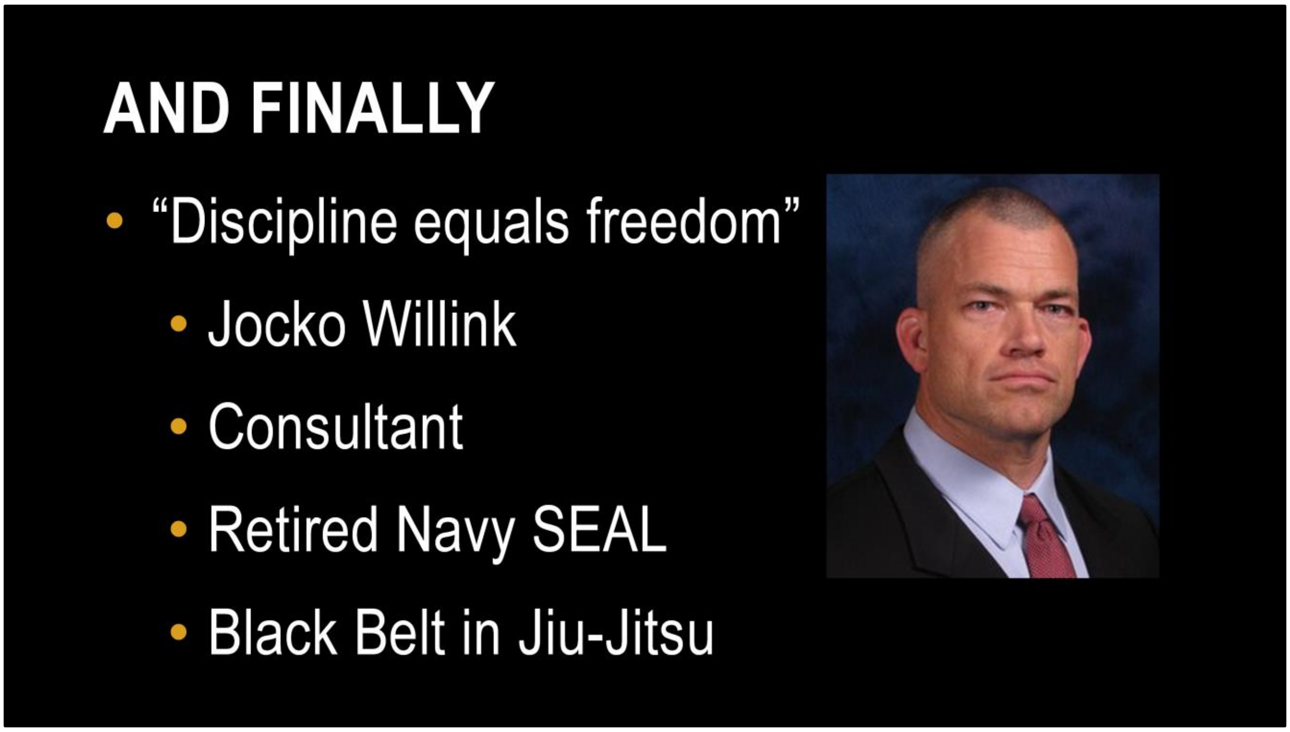
- We have an obligation to eliminate waste
- Not to curtail experimentation
- But to facilitate ***MORE*** of it
- And to make it ***MORE*** productive

AND FINALLY…



AND FINALLY

- Nothing I’ve talked about is all that complicated
- The hard part isn’t adopting these practices
- The hard part is maintaining discipline in their use
- Discipline is essential for seeing results



AND FINALLY

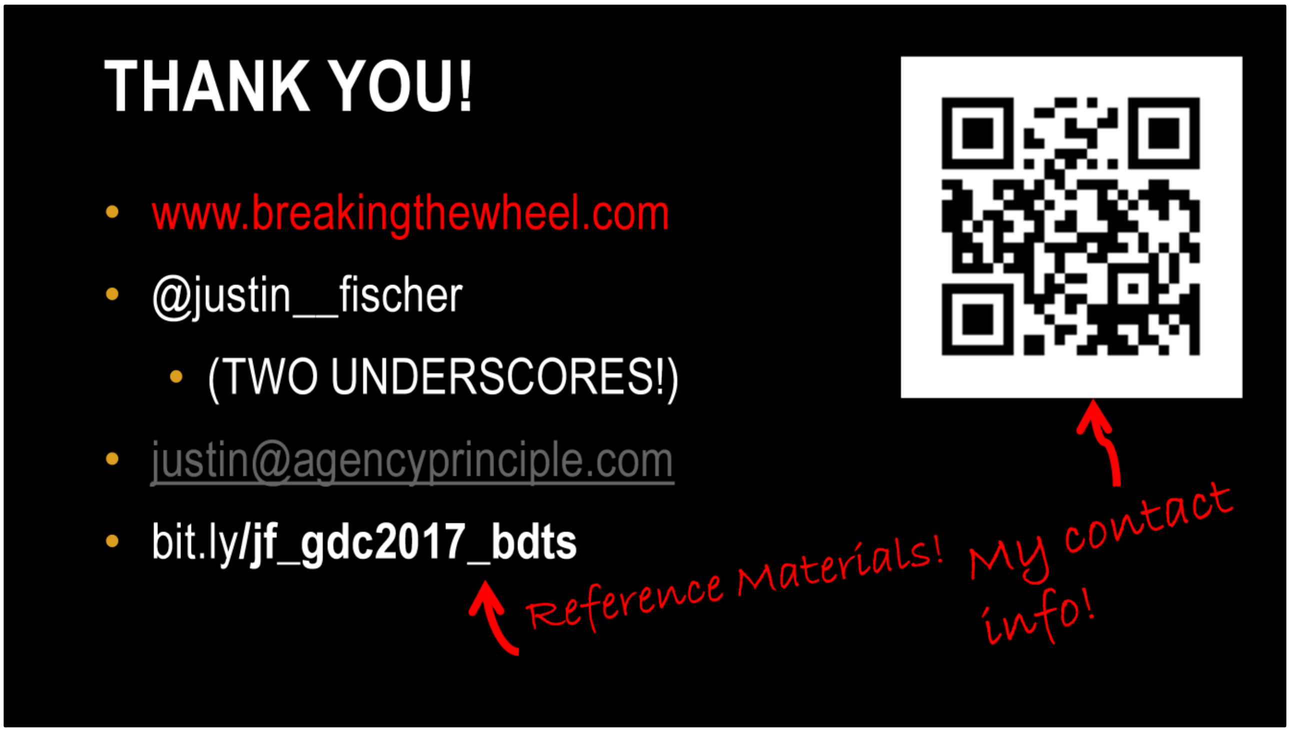
- "Discipline equals freedom"
  - Jocko Willink
  - Consultant
  - Retired Navy SEAL
  - Black Belt in Jiu-Jitsu

- If you have the discipline to eat a balanced diet and get regular exercise, you will have the physical freedom afforded by good health
- If you have the discipline to study consistently throughout the semester, you'll have the freedom to sleep the night before the final because you won't need to cram
- And…

AND FINALLY

- If you have the discipline to follow processes that reduce or eliminate waste
- You will have the freedom to spend your time generating more value for gamers instead of fighting fires

QUESTIONS?



THANK YOU!

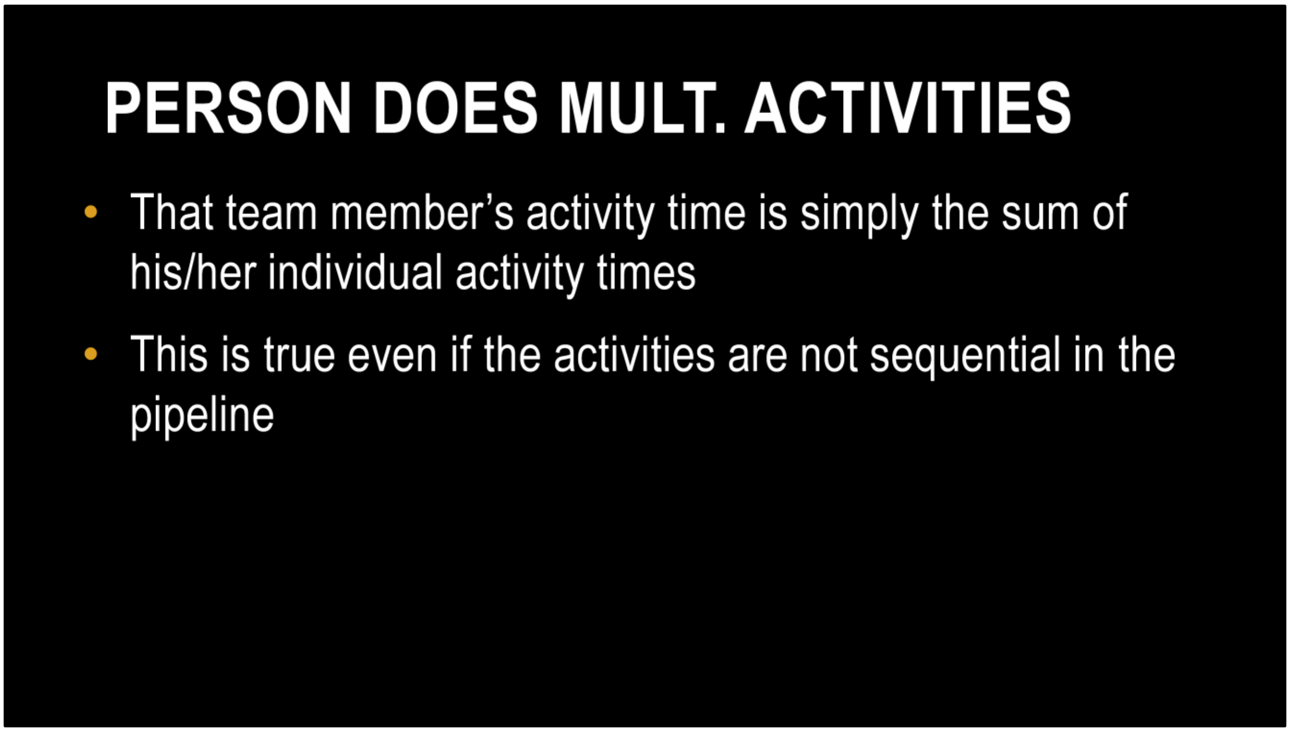
- www.breakingthewheel.com
- @justin__fischer
  - (TWO UNDERSCORES!)
- justin@agencyprinciple.com
- bit.ly/**jf_gdc2017_bdts**

Reference Materials!

My contact info!

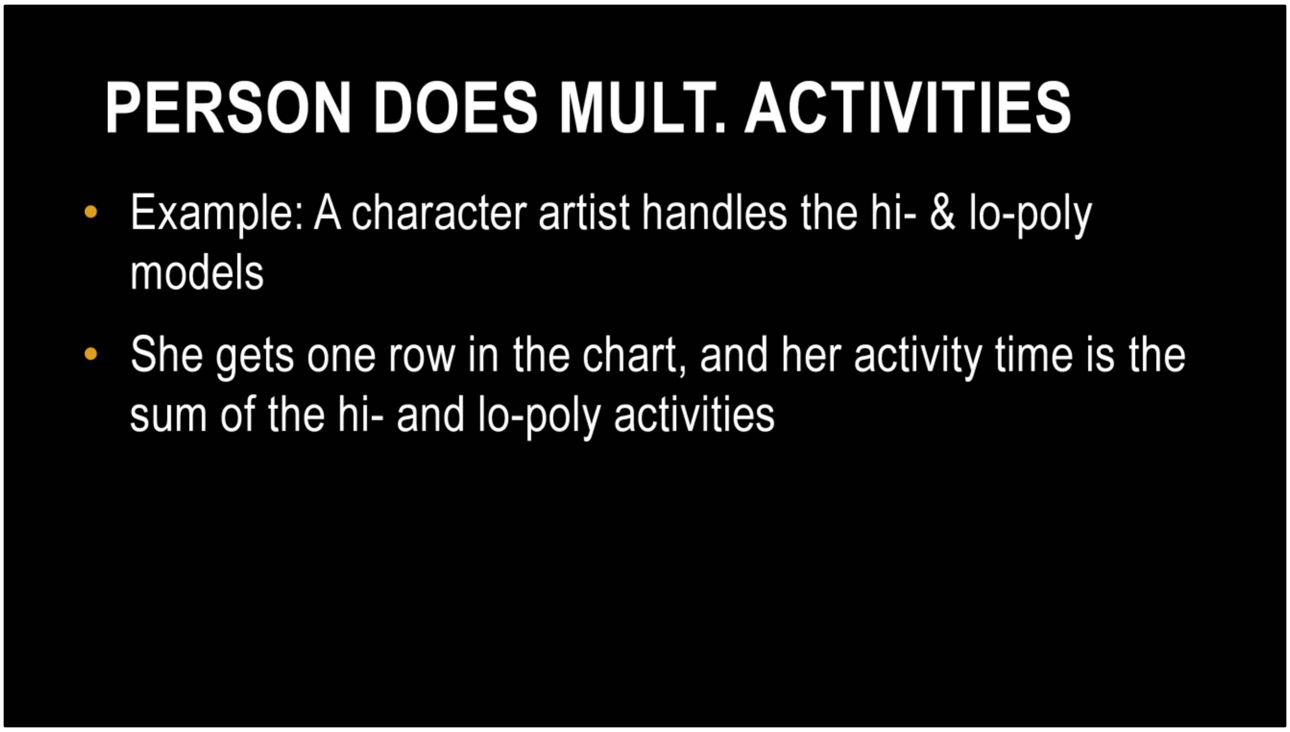
Feel free to hit me up by your vector of choice. I love talking shop!

APPENDIX: SPECIAL CASE CALCULATIONS

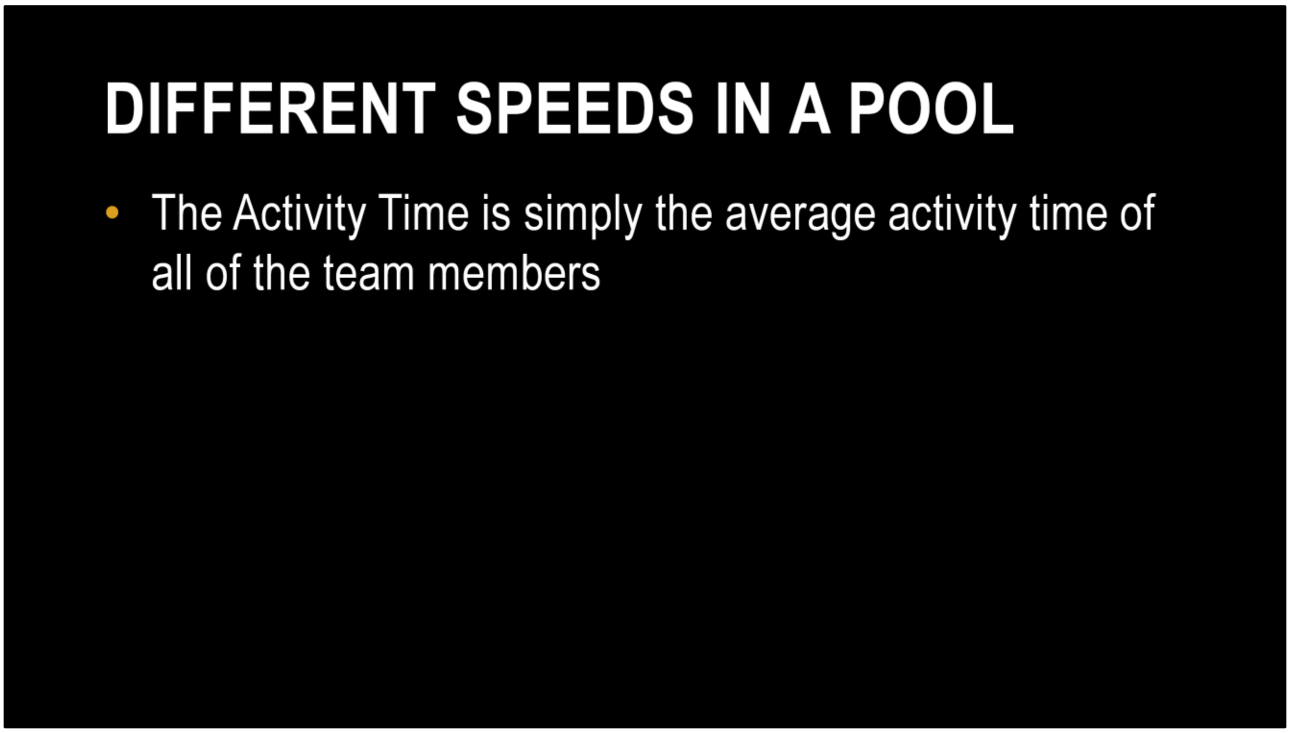


PERSON DOES MULT. ACTIVITIES

- That team member's activity time is simply the sum of his/her individual activity times
- This is true even if the activities are not sequential in the pipeline

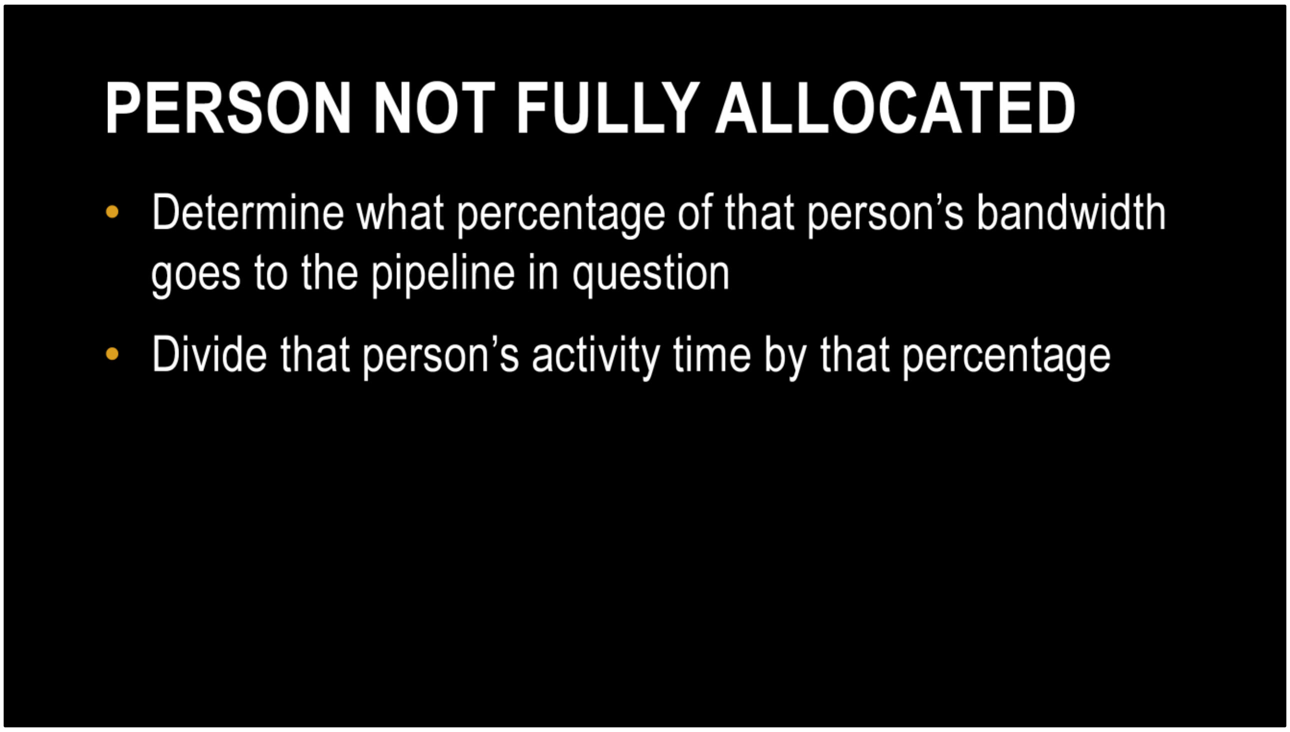

PERSON DOES MULT. ACTIVITIES

- Example: A character artist handles the hi- & lo-poly models
- She gets one row in the chart, and her activity time is the sum of the hi- and lo-poly activities



DIFFERENT SPEEDS IN A POOL

- The Activity Time is simply the average activity time of all of the team members

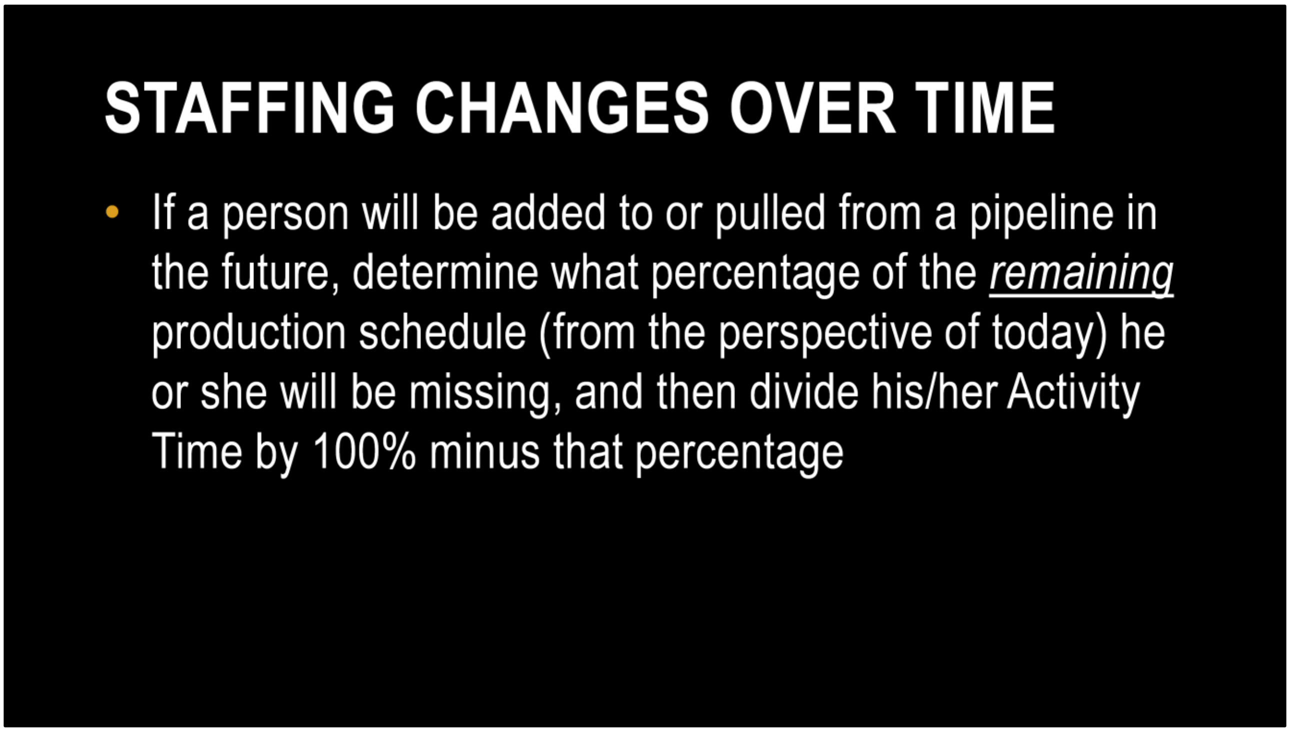


PERSON NOT FULLY ALLOCATED

- Determine what percentage of that person’s bandwidth goes to the pipeline in question
- Divide that person’s activity time by that percentage

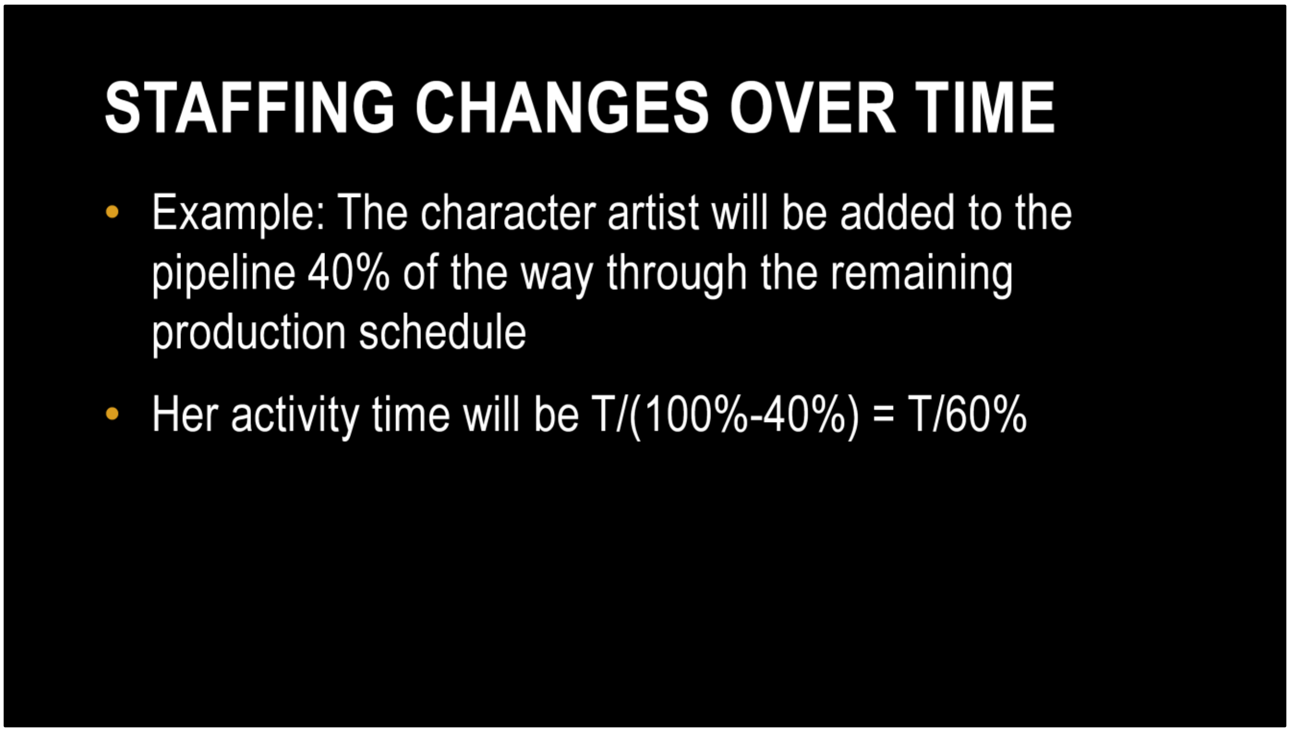
PERSON NOT FULLY ALLOCATED

- Example: The character artist also helps with environment modeling. 60% of her bandwidth is for character models. The other 40% goes to environment work.
- For the character art pipeline, her activity time is T/60%



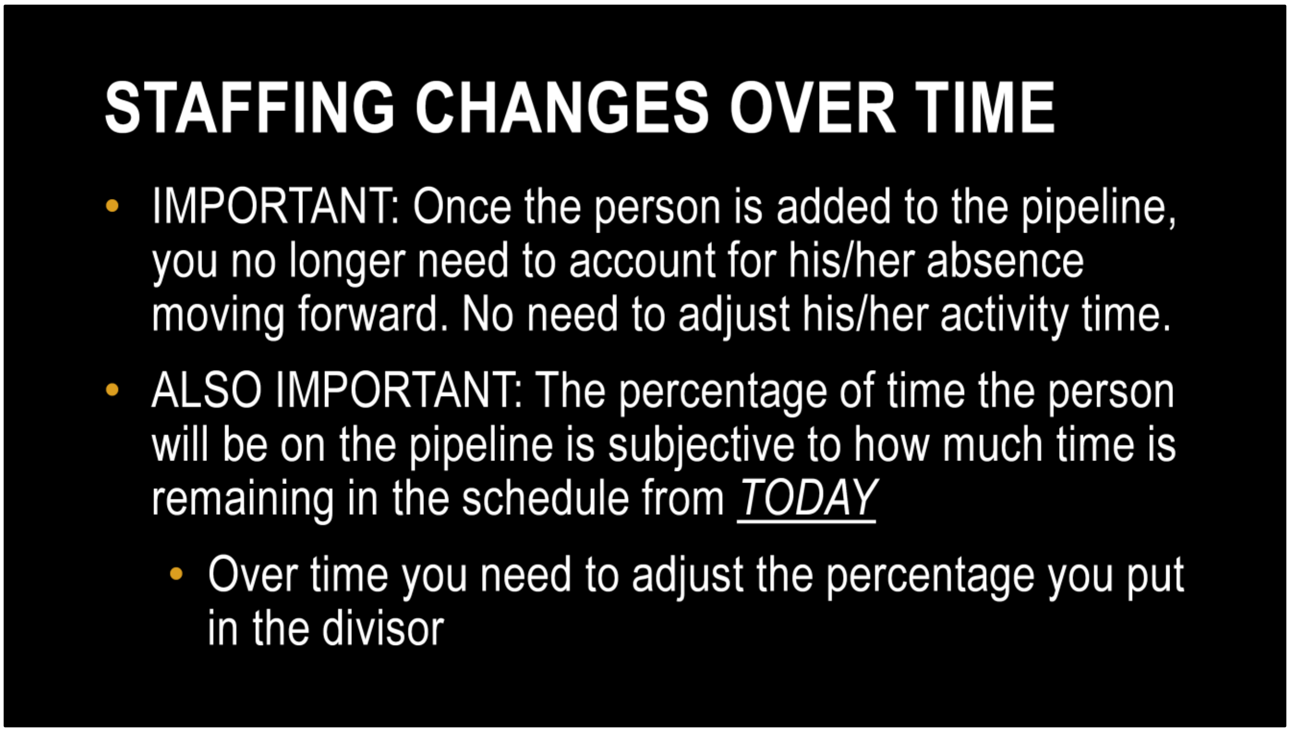
STAFFING CHANGES OVER TIME

- If a person will be added to or pulled from a pipeline in the future, determine what percentage of the ***remaining*** production schedule (from the perspective of today) he or she will be missing, and then divide his/her Activity Time by 100% minus that percentage



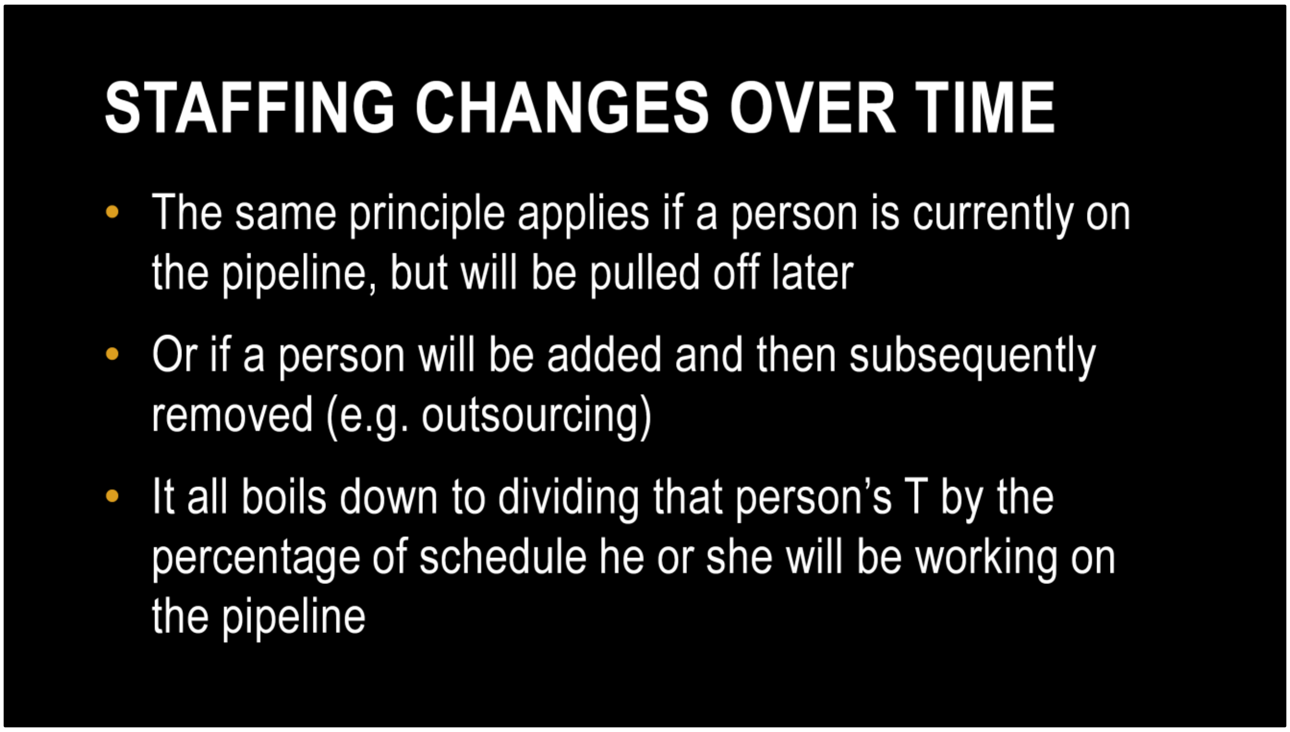
STAFFING CHANGES OVER TIME

- Example: The character artist will be added to the pipeline 40% of the way through the remaining production schedule
- Her activity time will be T/(100%-40%) = T/60%



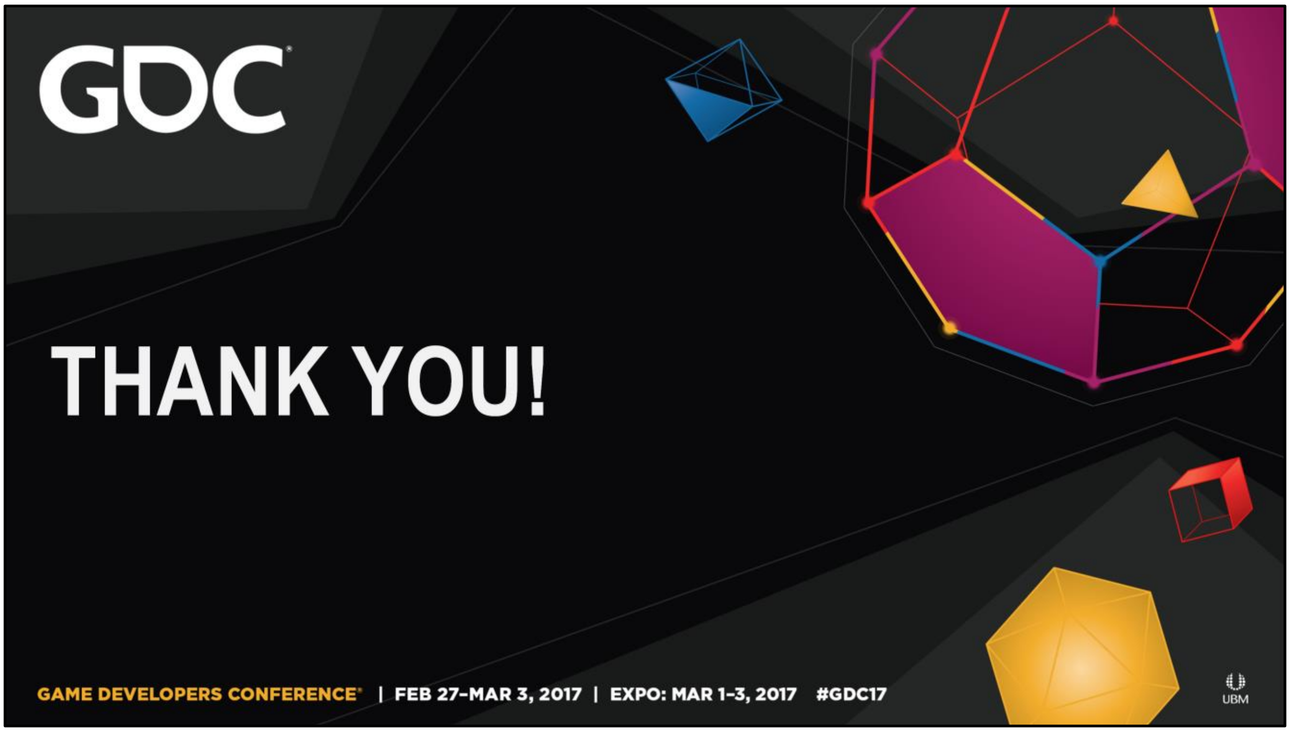
STAFFING CHANGES OVER TIME

- IMPORTANT: Once the person is added to the pipeline, you no longer need to account for his/her absence moving forward. No need to adjust his/her activity time.
- ALSO IMPORTANT: The percentage of time the person will be on the pipeline is subjective to how much time is remaining in the schedule from *TODAY*
  - Over time you need to adjust the percentage you put in the divisor



STAFFING CHANGES OVER TIME

- The same principle applies if a person is currently on the pipeline, but will be pulled off later
- Or if a person will be added and then subsequently removed (e.g. outsourcing)
- It all boils down to dividing that person's T by the percentage of schedule he or she will be working on the pipeline



GDC

THANK YOU!

GAME DEVELOPERS CONFERENCE® | FEB 27–MAR 3, 2017 | EXPO: MAR 1–3, 2017 #GDC17

UBM

Slide added so I know where to stop clicking!