INCREMENTAL ARCHITECTURE

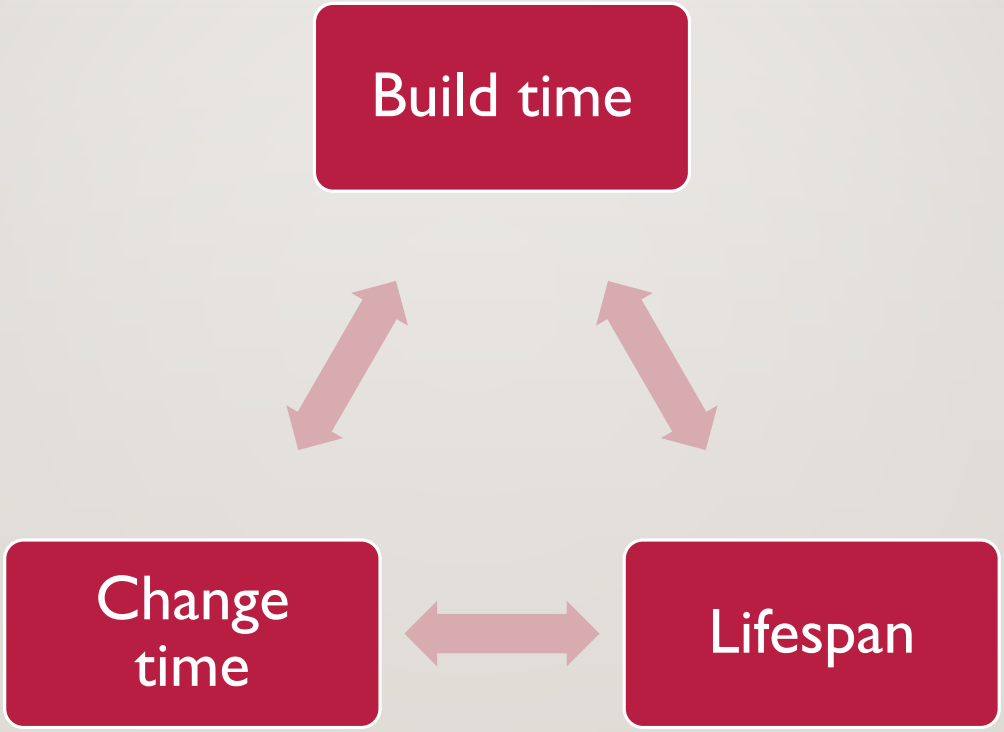
The architecture is only done when you unplug the last server.

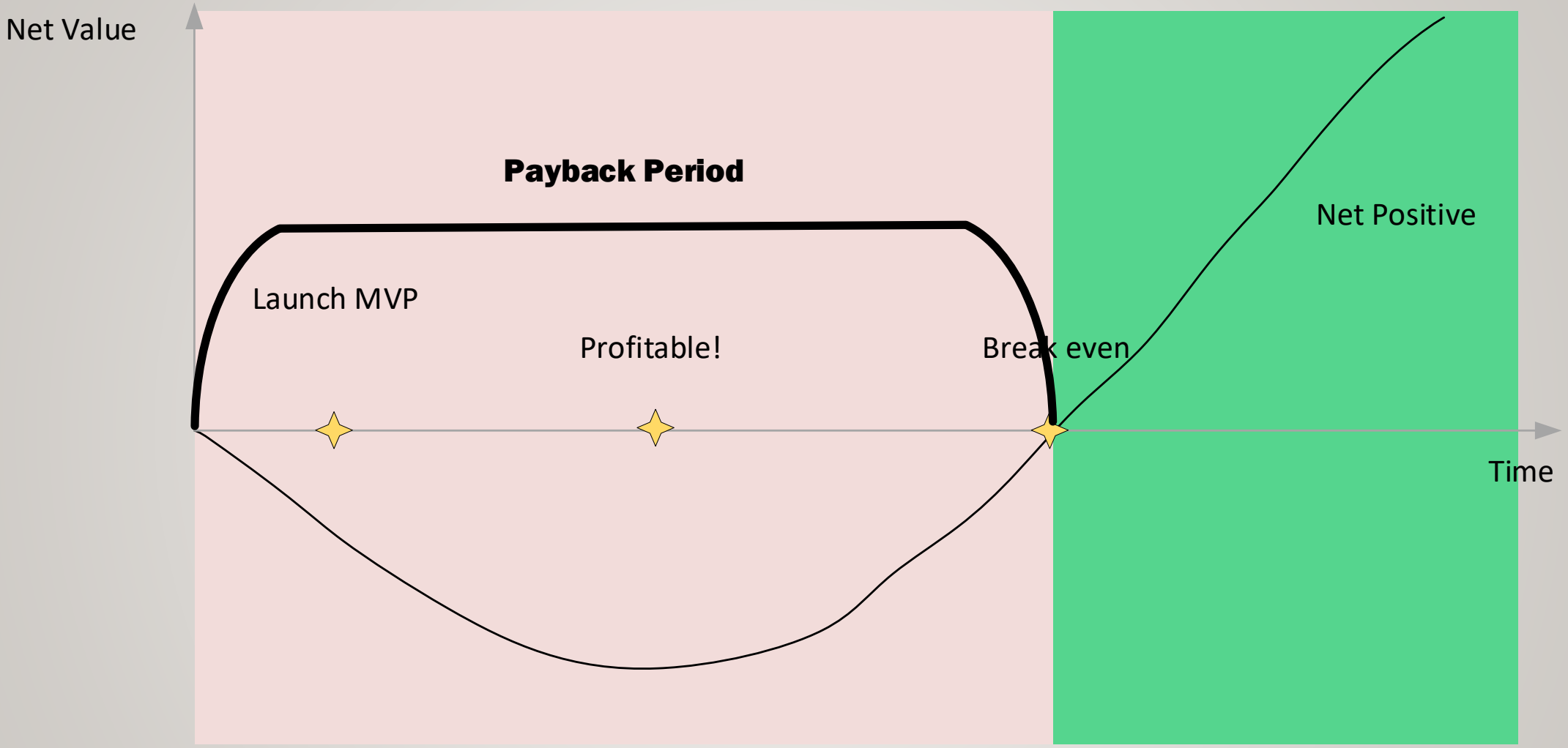
WHEN? HOW MUCH?

TIME IS FUNDAMENTAL

- Build time
- System lifespan
- Change time after launch

CONFLICTS IN TIME




Net Value
Payback Period
Net Positive
Launch MVP
Profitable!
Break even
Time

What limits system lifespan?

THE WORK

- The work of architecture must support timely delivery of a viable system.

EARLY WORK

- Business Goals
- Constraints
- Architecture Quality Ranking
- Architecturally Significant Requirements

ONGOING WORK

- Interface Table
- Architecture Quality Scenarios
- Internal Boundaries (and dispute resolution)
- Information Architecture
- Evaluations of the Architecture
- Verification the Implementations

Wherever possible, leave options open for the future.

HOW MUCH?

- Depends on project risk
- Risky projects get more attention
- Low risk gets less.

RISK FACTORS: TECHNOLOGY

- New language
- New deployment platform (new cloud, new OS, new mobile device)
- "Exotic" search, storage, security, or analytics tech
- Tech stack with highly general applicability. (No out-of-the-box architecture.)
- Long edit-compile-test cycle
- Integration with hardware that isn't abstracted away
- Components that cannot be replicated in development
- High barrier to deployment
- Creating a protocol that must be supported for multiple releases

RISK FACTORS: COMPLEXITY

- Architecture style other than centralized application or streaming data (e.g., mesh networked, peer-to-peer)
- High # integrations with external systems
- High throughput or availability requirements
- Soft or hard real-time requirements
- Active-active deployment to multiple locations
- Multiparty stateful interactions
- Tight resource constraints or high resource needs
- Presence of any of the following: ESB, SOA, SOAP, SAML, OAuth, IBM MQSeries.
- Presence of PCI Level 1 or 2, HIPAA, FDA, J/SOX, other compliance regimes
- Supporting a vertical industry standard

ACTIVITY: RISK SCORE FOR OUR SAMPLE SYSTEM

Which risk factors are present in our sample system?

ACTIVITY: RISK SCORE FOR OUR SAMPLE SYSTEM

Which risk factors are present in our sample system?

Hold up…

We haven't picked a deployment platform or tech stack.

ACTIVITY: RISK SCORE FOR OUR SAMPLE SYSTEM

- Which risk factors are present in our sample system?
- Assume AWS deployment with front end app in JS + React, back end in Go.
- Add up the total risk score

THINK ABOUT "SET-BASED CONCURRENT ENGINEERING"

"One of the ideas in lean product development is the notion of set-based concurrent engineering: considering a solution as the intersection of a number of feasible parts, rather than iterating on a bunch of individual "point-based" solutions. This lets several groups work at the same time, as they converge on a solution."

—Bill Wake, xp123.com

https://xp123.com/articles/set-based-concurrent-engineering/

See also https://xp123.com/articles/resources-on-set-based-design/

THINK ABOUT "SET-BASED CONCURRENT ENGINEERING"

Example from "Toyota's Principles of Set-Based Concurrent Engineering," Sobek, Ward, Liker, Sloan Management Review, 1999

Traditional "point-based" serial engineering

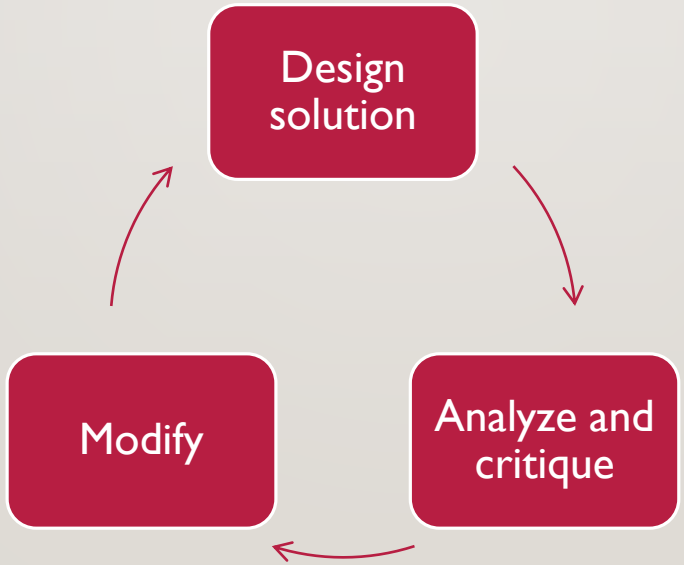
We would call this "waterfall" development

THINK ABOUT "SET-BASED CONCURRENT ENGINEERING"

Example from "Toyota's Principles of Set-Based Concurrent Engineering," Sobek, Ward, Liker, Sloan Management Review, 1999

"Point-based" concurrent engineering

Cross-functional team:
Marketing
Body
Chassis
Manufacturing



"Agile" or "iterative" development

THINK ABOUT "SET-BASED CONCURRENT ENGINEERING"

Example from "Toyota's Principles of Set-Based Concurrent Engineering," Sobek, Ward, Liker, Sloan Management Review, 1999

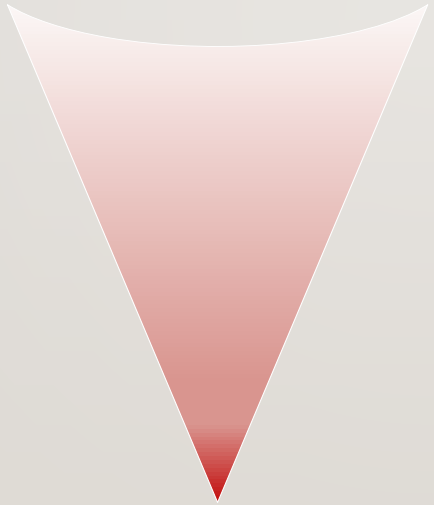
Design Eng.

Several designs meet our requirements

Working within those limits, these are the possible designs

This is nearly final, please review.

Final design



Manufacturing Eng.

Our capabilities are best suited to these designs

We can order tool steel and start planning the line layout

We'd like these changes to optimize mfg cost, then we can order castings

We'll make the final tools and start pilot

EXAMPLE IN OUR SPACE

Architecture

We think a message bus is needed to decouple the components

Our latency and uptime requirements look like this. Therefore we're down to A & B

We think A will work. Here's the topology, queues, and volume.

Final design

Operations

We have support agreements with X, Y, and Z that might fit your needs.

We will install a monitoring backend that can support either one.

We agree that will work, but need some tweaks to topology to support our network.

We'll install the infrastructure and plug in monitoring.

A SERIES OF INCREMENTAL DECISIONS

1. "We need messaging."
2. "We need at least once delivery with allowed downtime of an hour on the subscriber."
3. Rabbit MQ vs Active MQ vs Hornet vs IBM MQ Series, etc.
4. Topic names, hierarchy, message format, encoding, relays

FEATURES OF SET-BASED CONCURRENT ENGINEERING

- More models, prototypes, and proofs-of-concept.
- Common to have multiple full-scale models at the same time.
- Slow progression from rough design to detailed design.
- Avoid premature commitment.
- Desire to avoid backtracking on commitments.

“THE LAST RESPONSIBLE MOMENT”

Leave decisions open until:

Cost of decision + switching cost < cost of working around lack of decision

MATCH DEVELOPMENT METHOD TO ENGINEERING STYLE

Method	Engineering Style
Classic waterfall	Point-based serial
Iterative	Point-based concurrent
XP	Point-based concurrent
Scrum	Point-based concurrent
Agile	Point-based concurrent
RUP	Set-based concurrent
Kanban	N/A – Kanban is a workload management tool, not a development method!

MARKETABLE FEATURES & ARCHITECTURAL ELEMENTS

We don’t need to “do” all the architecture up front.

MINIMUM MARKETABLE FEATURE (MMF)

- Distinct and deliverable feature of the system.
- Observable to the user.
- Provides significant value to the customer.
- "Self-contained," can be delivered without other features.
- Smallest possible realization of that feature.

From "Software by Numbers", Denne & Cleland-Huang

OUR SYSTEM: MMF OR NOT?

Deliverable	MMF?
Comprehensive automated test suite	No – users don't care about tests
Class library for integrating with 3rd party cards	No – not user visible
Redis-backed job queue	Definitely not. Might be anti-feature if adds delay
Message bus	No
Renew early & save program	Yes!
CSR screen for rebilling an account	Yes!
Loyalty card integration	Maybe – is it visible? Is it minimal?

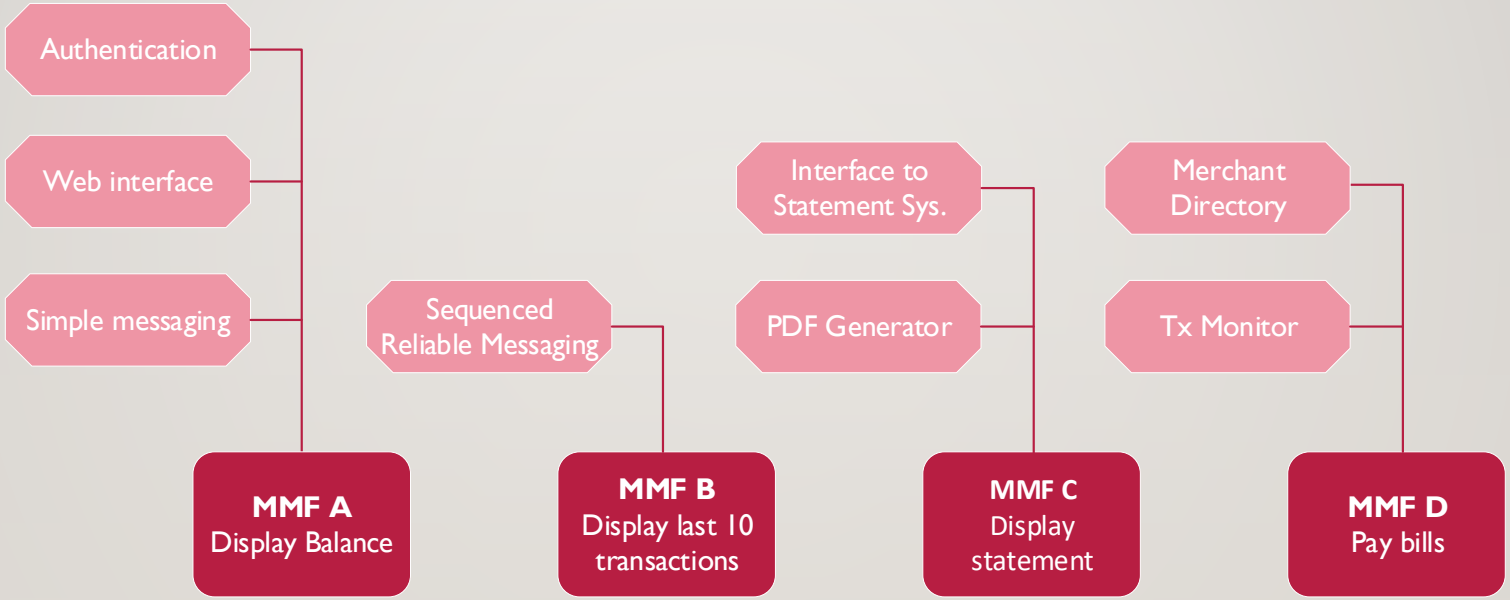
DECOMPOSE SYSTEMS BY MMFs

- Enable iterative development and rollout
- Realize benefits sooner
- Learn from production sooner
- Match investment to benefit at fine grain
- Avoid "Second System" and "Ivory Tower" syndromes

ARCHITECTURE ELEMENT (AE)

- Underlying support
- Involves hardware, software, or network components
- Allows delivery of one or more MMFs
- Purely cost elements. Do not deliver value on their own.

MMFs PULL AEs



BENEFITS OF RECOGNIZING AEs

- Makes architecture visible to stakeholders

BENEFITS OF RECOGNIZING AEs

- Makes architecture visible to stakeholders
- Clearly shows dependencies

BENEFITS OF RECOGNIZING AEs

- Makes architecture visible to stakeholders
- Clearly shows dependencies
- Allows incremental architecture

ACTIVITY: AEs NEEDED

- Some MMFs for our sample system:
  - MMF P: Vendor can list a single service
  - MMF T: Customer can autorenew
  - MMF V: We notify customers if their credit card is near expiration

In chat, name some architecture elements we need for each MMF.

Example
Suppose we had MMF A "Delivery occurs with nanosecond precision", you might say
A: atomic clock, ballistic missile guidance system, GPS-enabled terriers

SEQUENCING FEATURES

WHAT ORDER SHOULD WE DO THESE JOBS IN?

Job	Estimate
A	10 weeks
B	5 weeks
C	2 weeks

WHAT ORDER SHOULD WE DO THESE JOBS IN?

Sequence	All jobs finished in
A, B, C	17 weeks
A, C, B	17 weeks
B, C, A	17 weeks
B, A, C	17 weeks
C, A, B	17 weeks
C, B, A	17 weeks

WHAT ORDER SHOULD WE DO THESE JOBS IN?

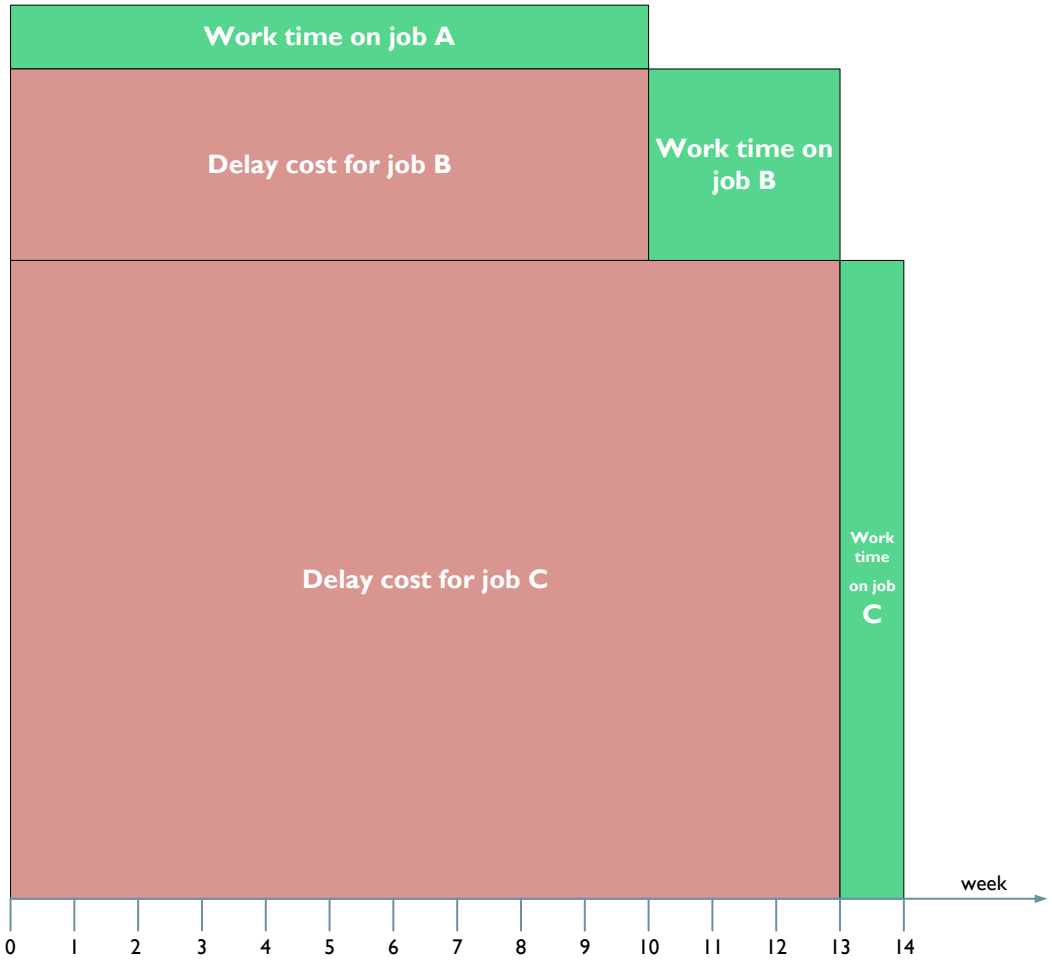
Sequence	All jobs finished in	Average Wait Time
A, B, C	17 weeks	14 weeks
A, C, B	17 weeks	13 weeks
B, C, A	17 weeks	9.6 weeks
B, A, C	17 weeks	12.3 weeks
C, A, B	17 weeks	9.3 weeks
C, B, A	17 weeks	8.6 weeks

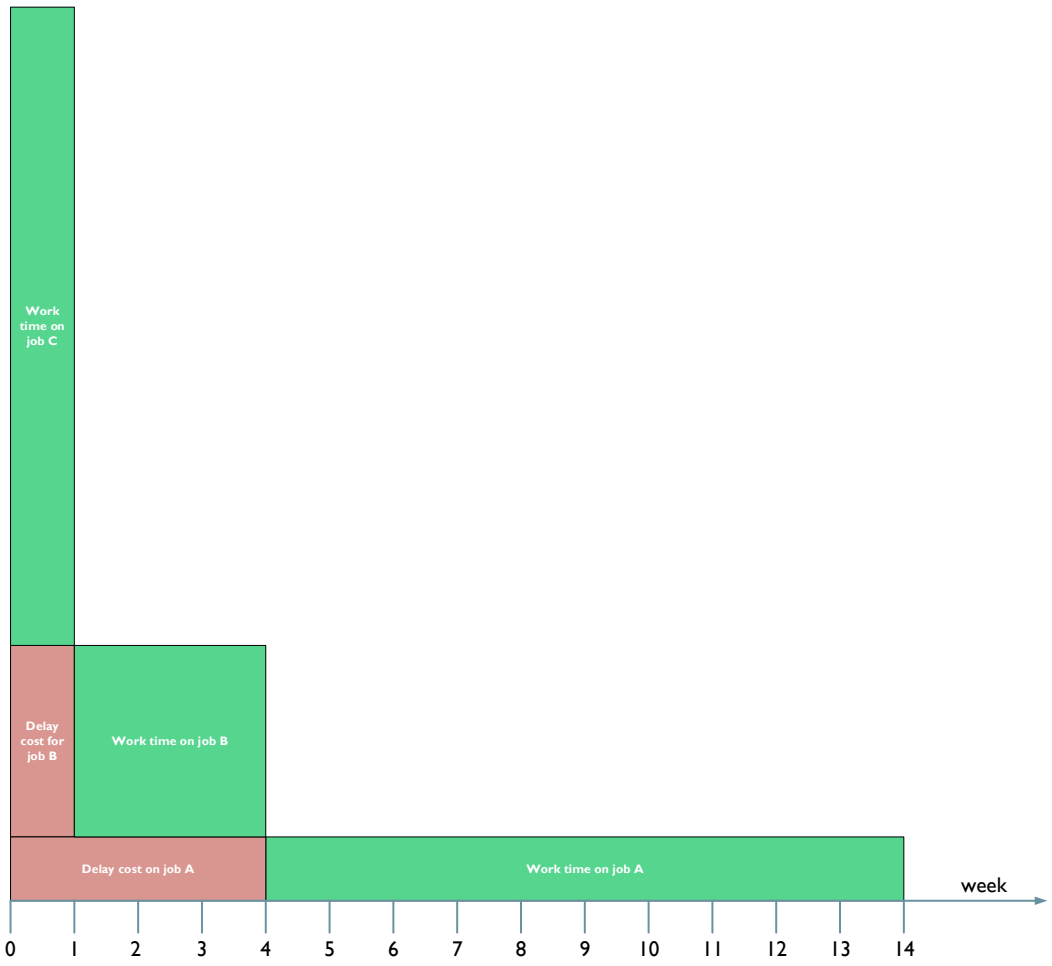
WHAT ORDER SHOULD WE DO THESE JOBS IN?

Sequence	All jobs finished in	Average Wait Time
A, B, C	17 weeks	14 weeks
A, C, B	17 weeks	13 weeks
B, C, A	17 weeks	9.6 weeks
B, A, C	17 weeks	12.3 weeks
C, A, B	17 weeks	9.3 weeks
C, B, A	**17 weeks**	**8.6 weeks**

WHAT ORDER SHOULD WE DO THESE JOBS IN?

Job	Estimate	Cost of Delay
A	10 weeks	1
B	3 weeks	3
C	2 weeks	10


Work time on job A
Delay cost for job B
Work time on job B
Delay cost for job C
Work time on job C
week
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14


Work time on job C
Delay cost for job B
Work time on job B
Delay cost on job A
Work time on job A
week
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

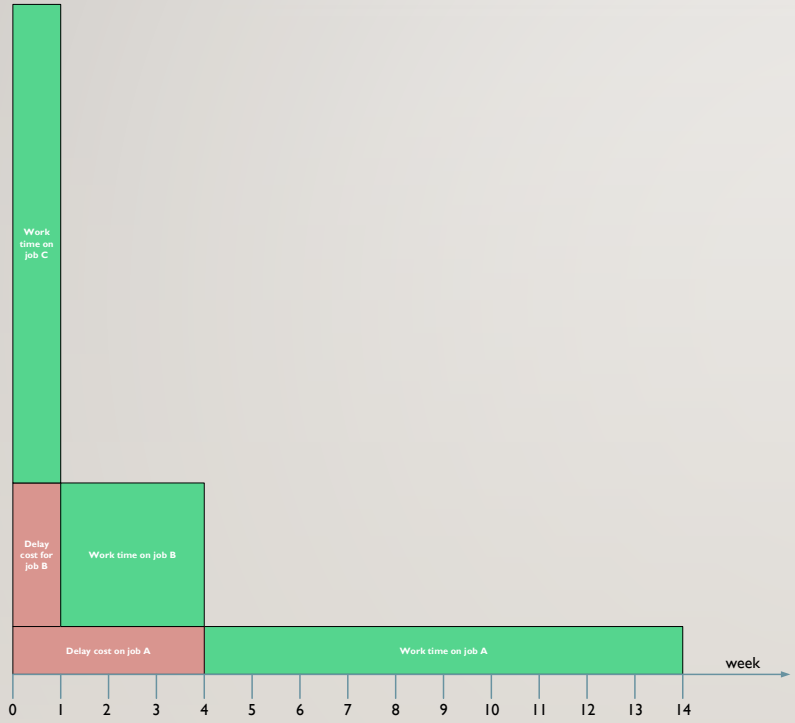
WEIGHTED SHORTEST JOB FIRST

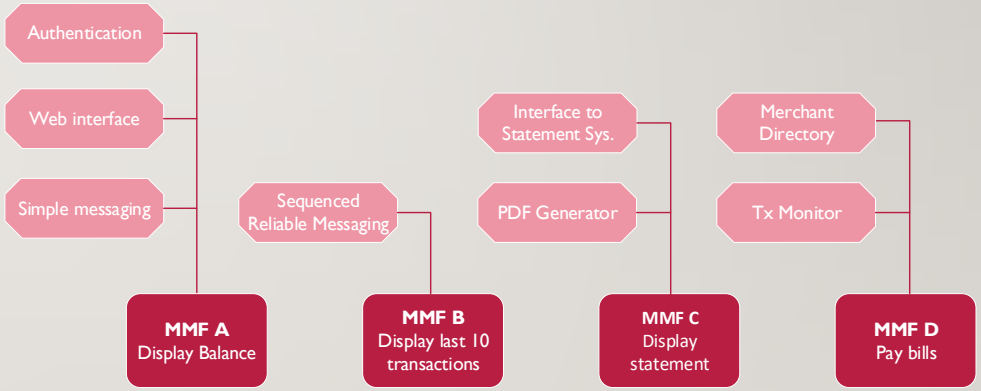
Weight = Cost / Duration

Job	Estimate	Cost of Delay	Weight
A	10 weeks	1	0.1
B	3 weeks	3	1
C	2 weeks	10	10

Pick the highest weight next

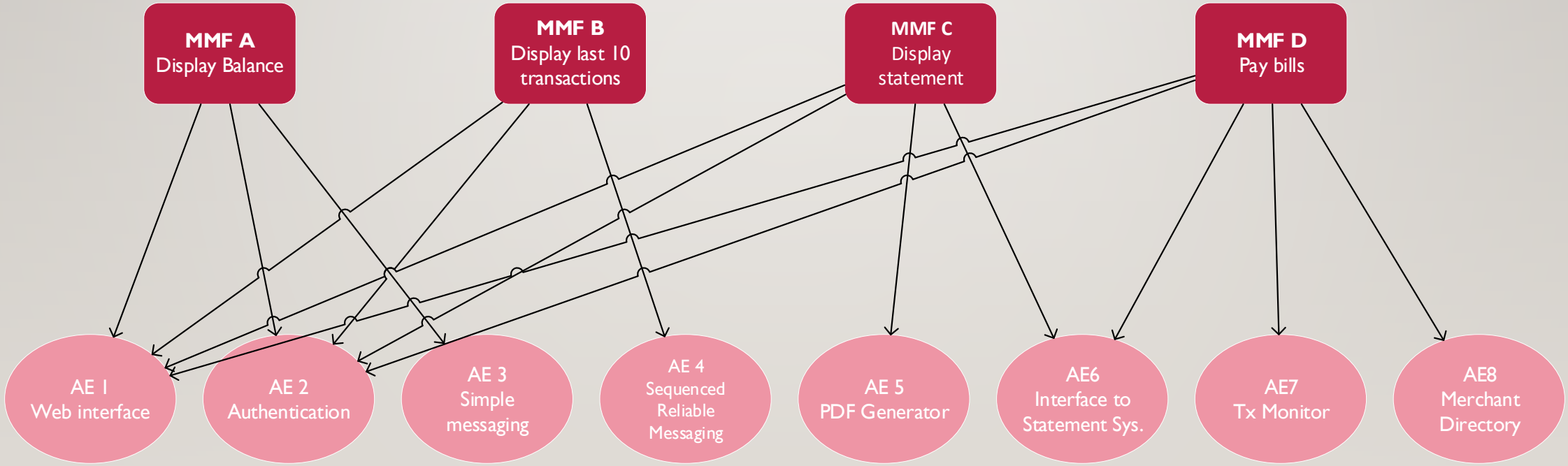
PUTTING IT TOGETHER: WSJF + MMF/AE





PUTTING IT TOGETHER: WSJF + MMF/AE

- Both duration and cost of delay depend on the sequence



PUTTING IT TOGETHER: WSJF + MMF/AE

- Both duration and cost of delay depend on the sequence

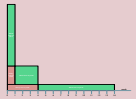
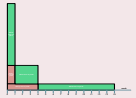
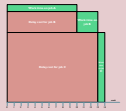
1st	2nd	3rd	4th
MMF A AE 1, 2, 3	MMF B AE 4	MMF C AE 5, 6	MMF D AE 7, 8

PUTTING IT TOGETHER: WSJF + MMF/AE

- Both duration and cost of delay depend on the sequence

1st	2nd	3rd	4th
MMF A AE 1, 2, 3	MMF B AE 4	MMF C AE 5, 6	MMF D AE 7, 8
MMF B AE 1, 2, 4	MMF A AE 3	MMF C AE 5, 6	MMF D AE 7, 8
MMF C AE 1, 2, 5, 6	MMF D AE 7, 8	MMF B AE 4	MMF A AE 3

APPLY WSJF TO EACH POSSIBLE SEQUENCE

1st	2nd	3rd	4th	WSJF
MMF A AE 1, 2, 3	MMF B AE 4	MMF C AE 5, 6	MMF D AE 7, 8	
MMF B AE 1, 2, 4	MMF A AE 3	MMF C AE 5, 6	MMF D AE 7, 8	
MMF C AE 1, 2, 5, 6	MMF D AE 7, 8	MMF B AE 4	MMF A AE 3	

YAGNI OR NOT?

YOU AIN'T GONNA NEED IT

"Always implement things when you actually need them, never when you just foresee that you need them."

-- Ron Jeffries

https://ronjeffries.com/xprog/articles/practices/pracnotneed/

WHAT ENABLES YAGNI?

1. Collective code ownership.
2. Merciless refactoring.
3. Comprehensive unit tests.

ACROSS PROCESS BOUNDARIES, WEAKEN YAGNI

- Design interfaces to be slightly more general.

EXAMPLE: HATS ON LOBSTERS

Hats

Request Hat

A hat is a formal, verified, way of posting a comment while speaking for a project, organization, or company. Each user may have multiple hats, one of which may be selected to be worn when posting a comment or sending a private message.

User	Hat	Link
stsp	Apache Subversion developer	https://people.apache.org/committer-index.html#stsp
JonLuca	Apple Employee	jonluca@apple.com
zg	Apple Employee	zzg@apple.com
crazyloglad	Arcan developer	https://arcan-fe.com
jelly	Arch Linux Developer	https://www.archlinux.org/people/developers/#jelle

Hat: XYZ Project Member

Link: user@project.org, or a URL to an employment page

Comment:

Will only be shown to moderators during approval

Request Hat

HATS ARE THE FEATURE, NOT THE SERVICE

SLIGHTLY MORE GENERAL

- Not a “Hat Request”… just a “Request”
- Approval by another party is a common process
- Build a service around the process not the data