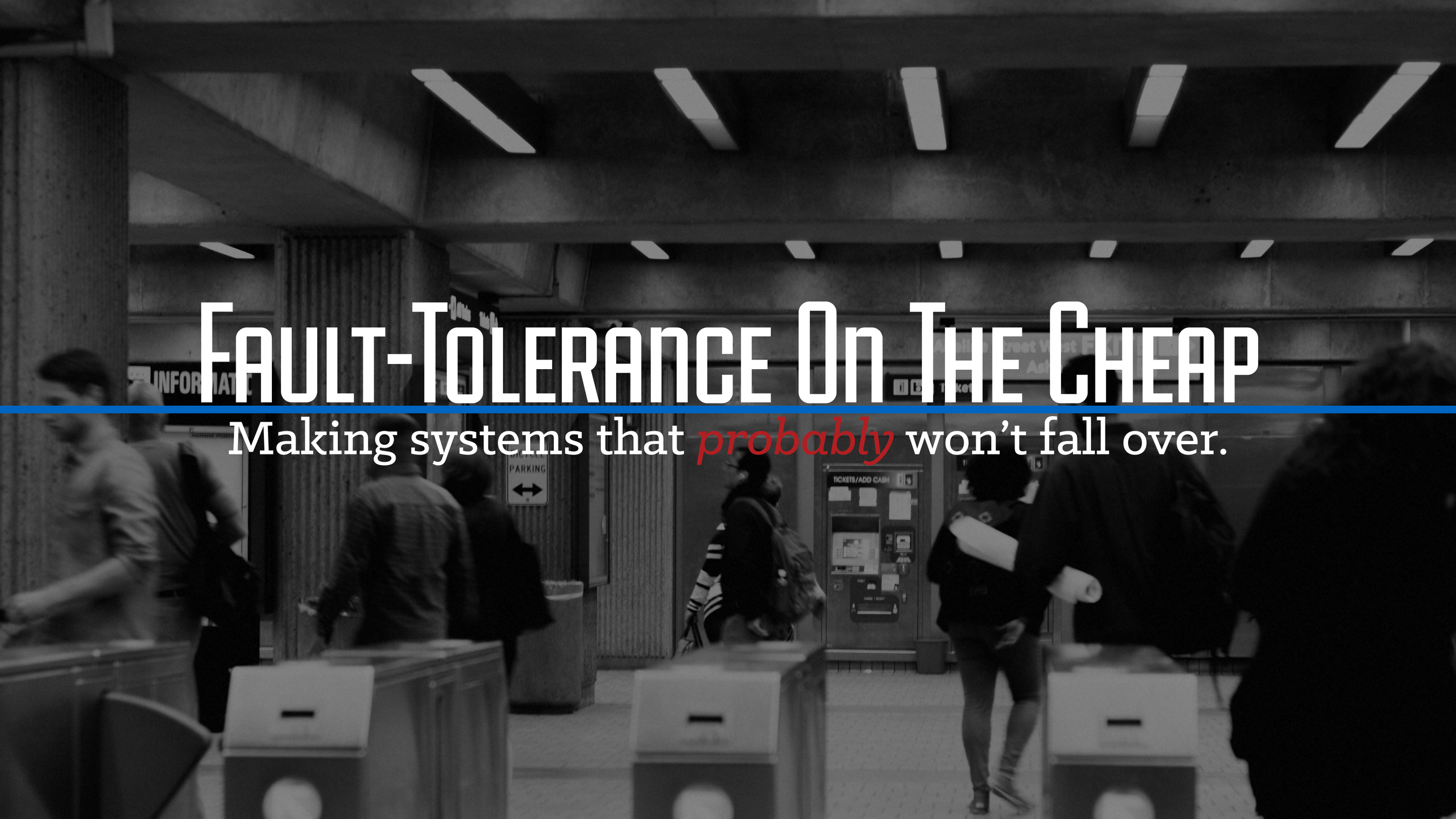


Fault-Tolerance On The Cheap

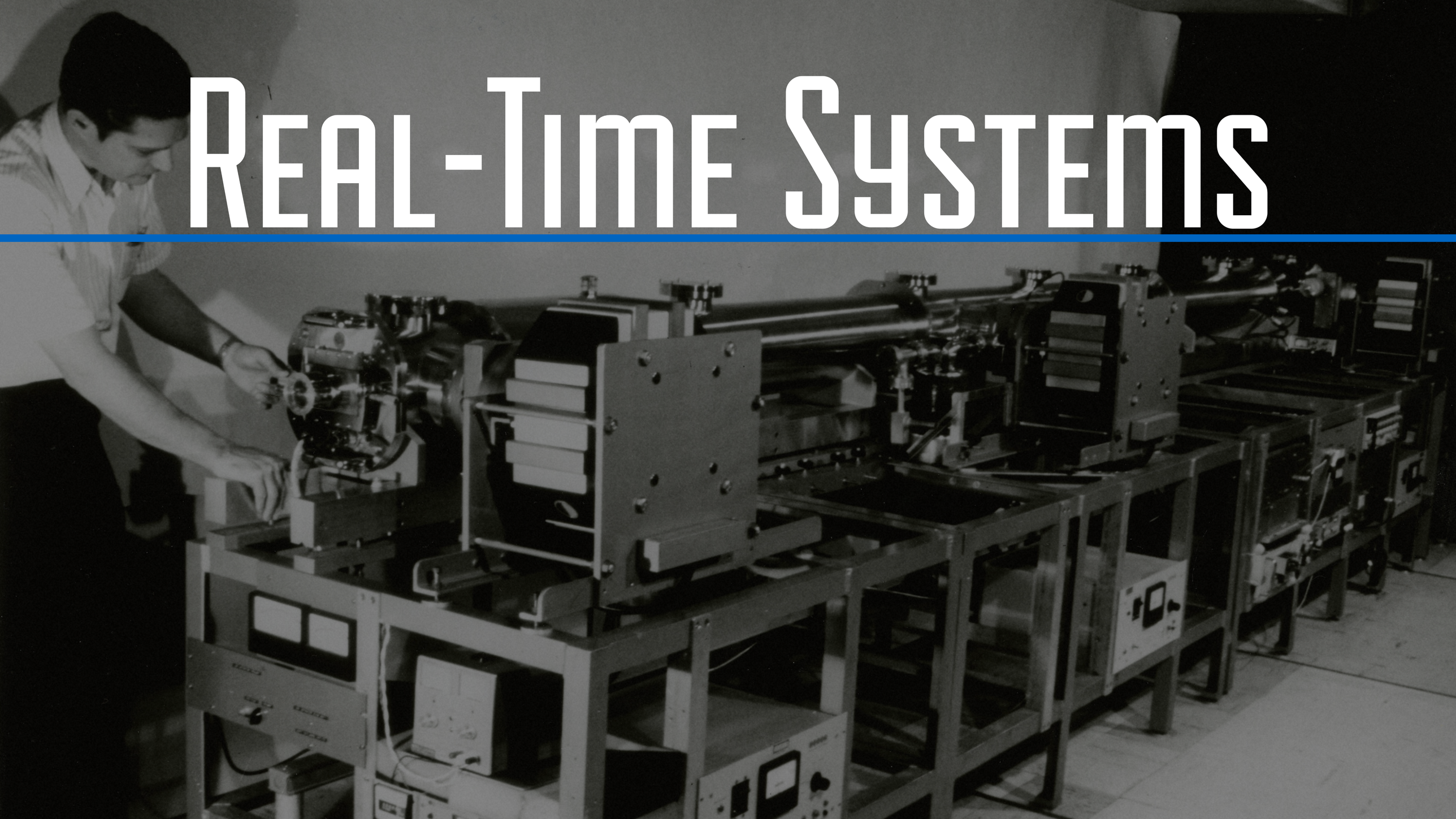
Making systems that *probably* won't fall over.

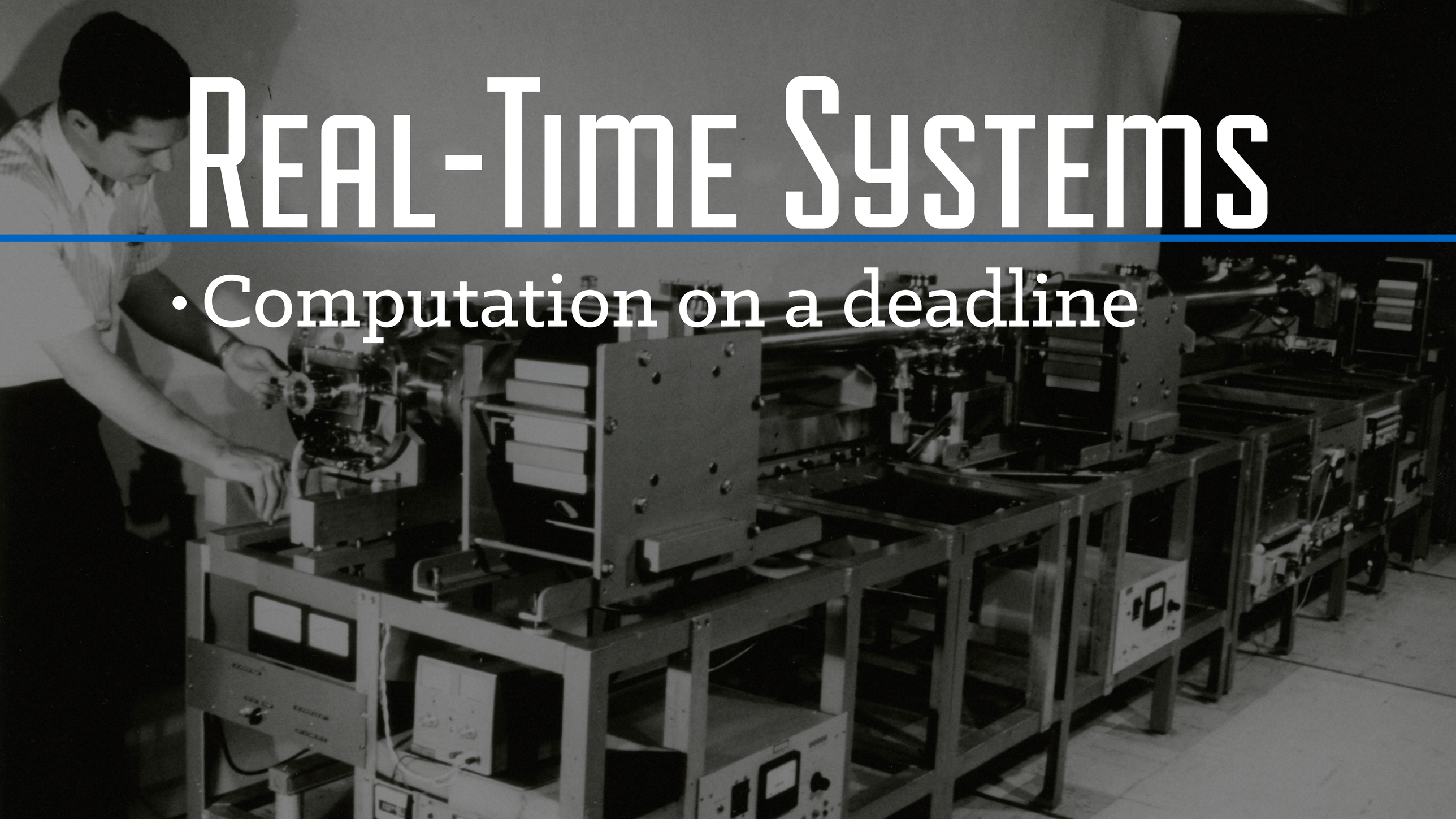
Hi, folks!

I do things to/with computers.

I’m a real-time, networked systems engineer.

Real-Time Systems



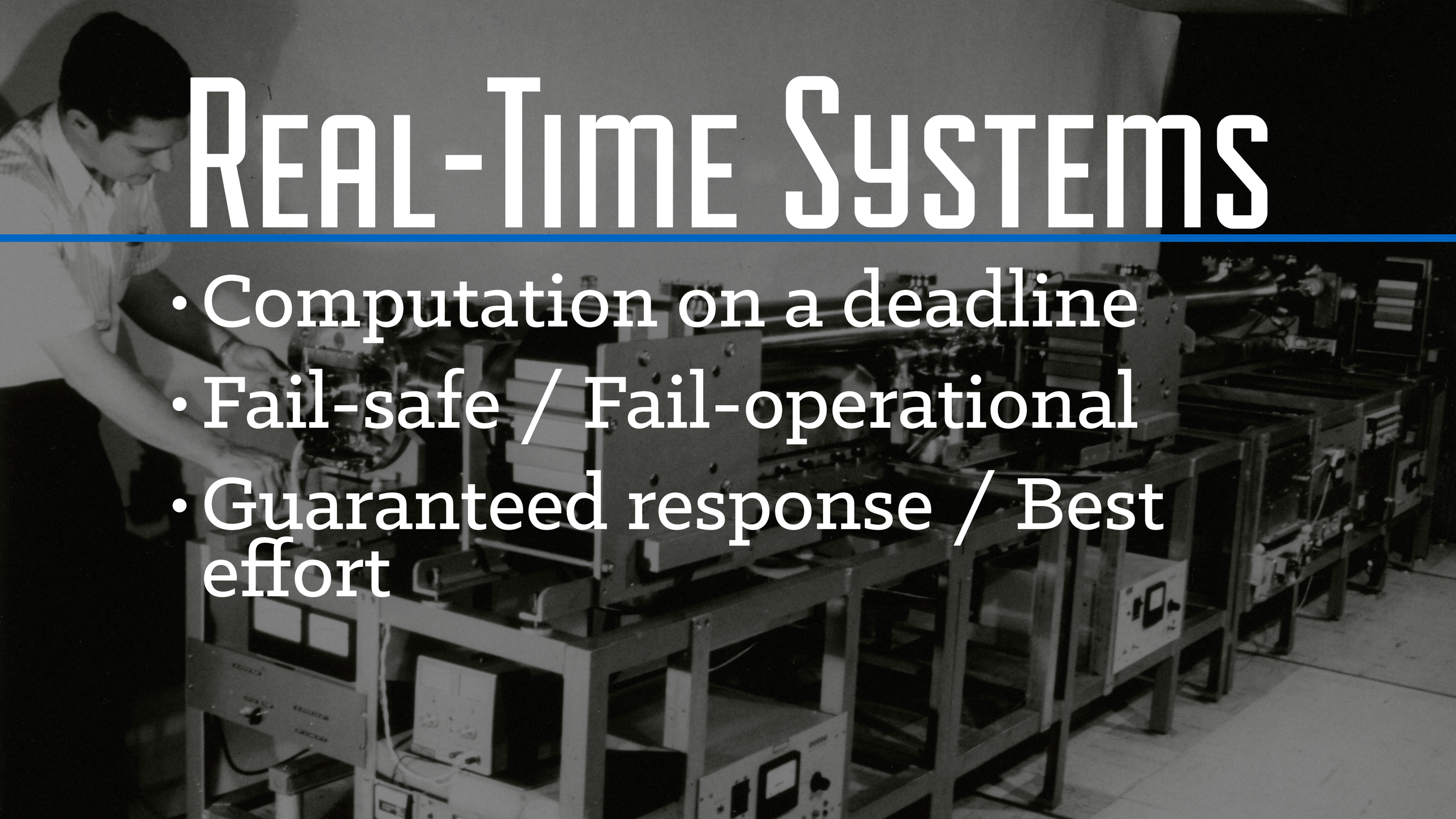


Real-Time Systems

- Computation on a deadline

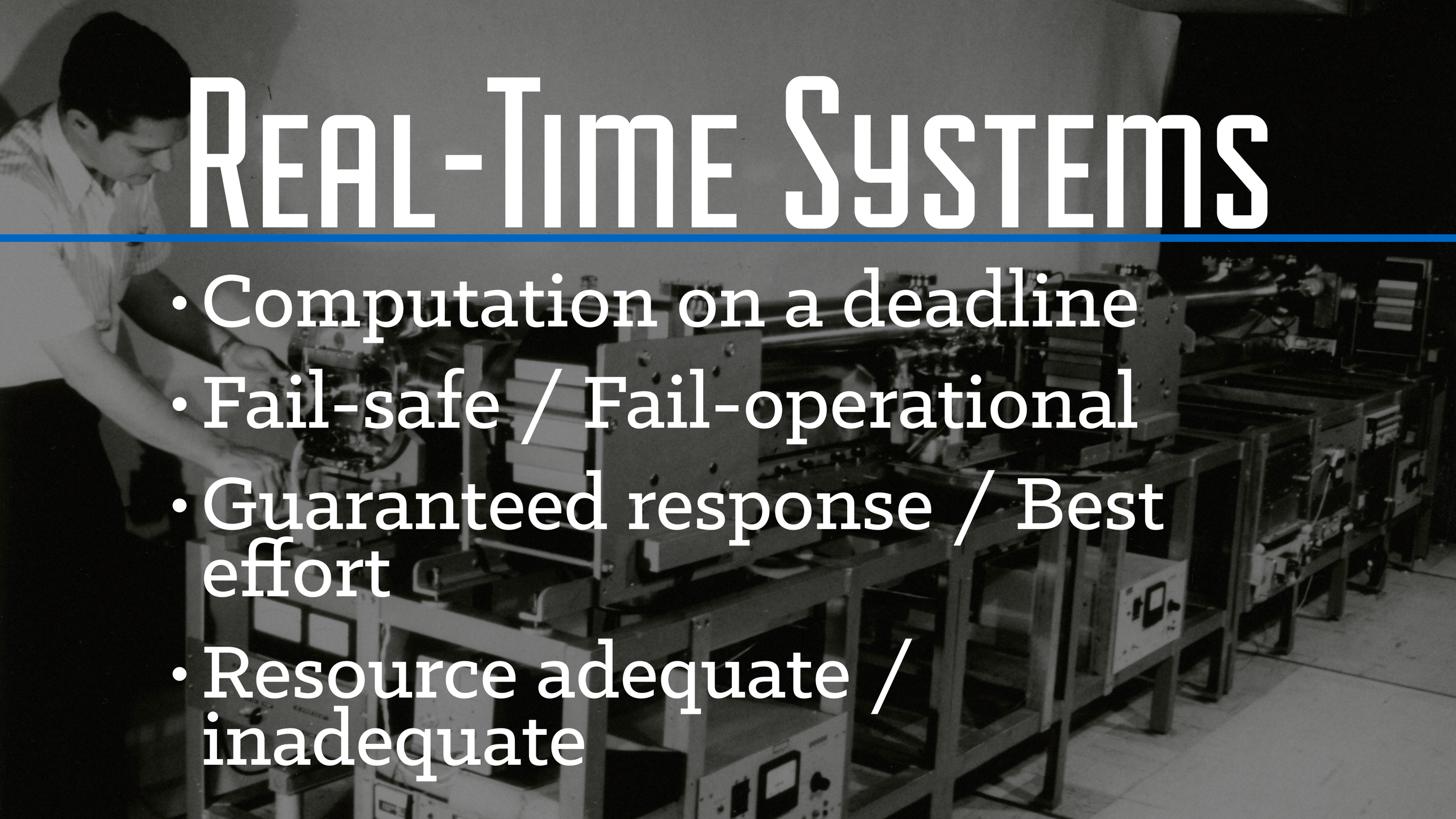
Real-Time Systems

- Computation on a deadline
- Fail-safe / Fail-operational



Real-Time Systems

- Computation on a deadline
- Fail-safe / Fail-operational
- Guaranteed response / Best effort



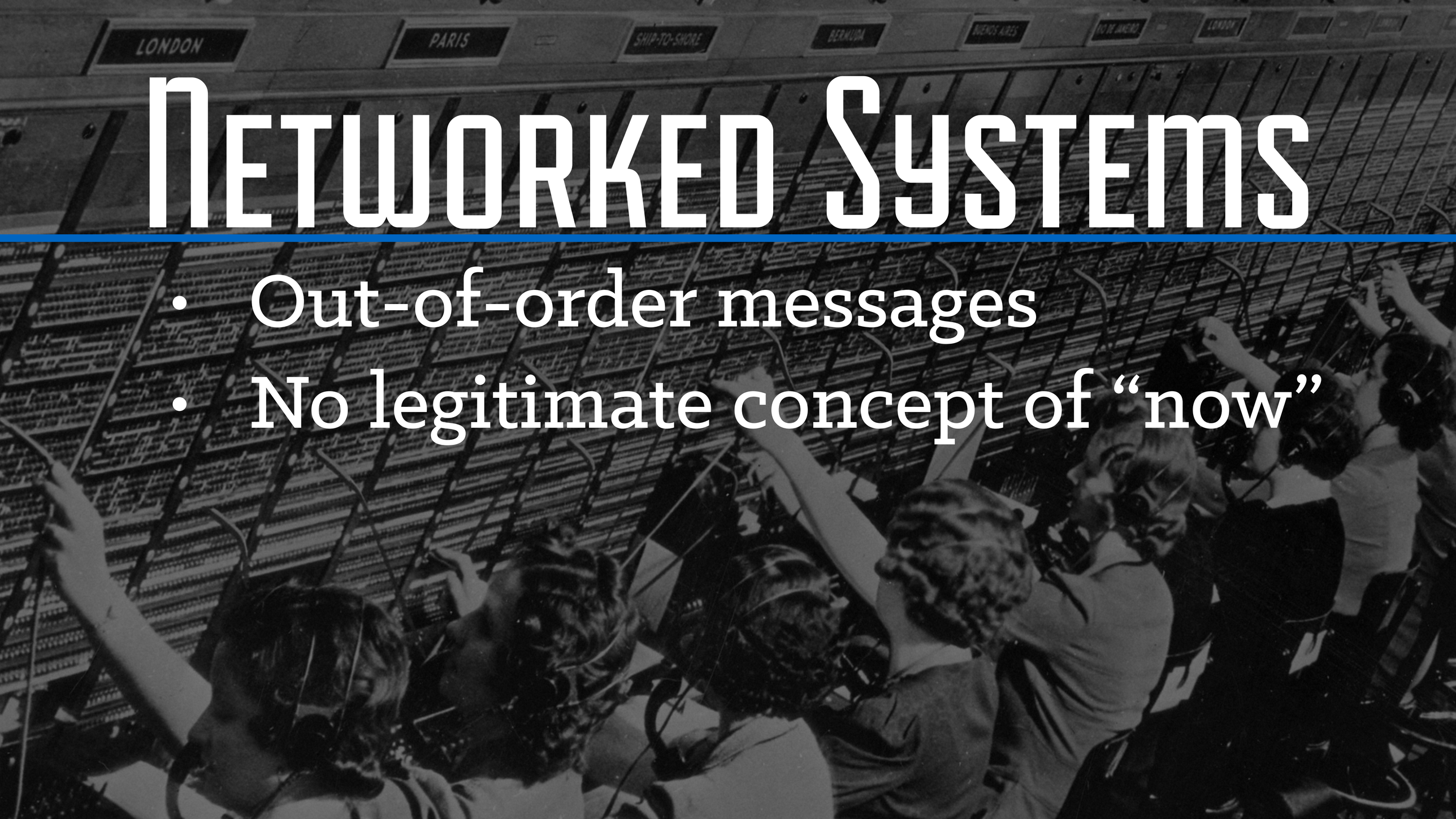
Real-Time Systems

- Computation on a deadline
- Fail-safe / Fail-operational
- Guaranteed response / Best effort
- Resource adequate / inadequate

Networked Systems

Networked Systems

- Out-of-order messages

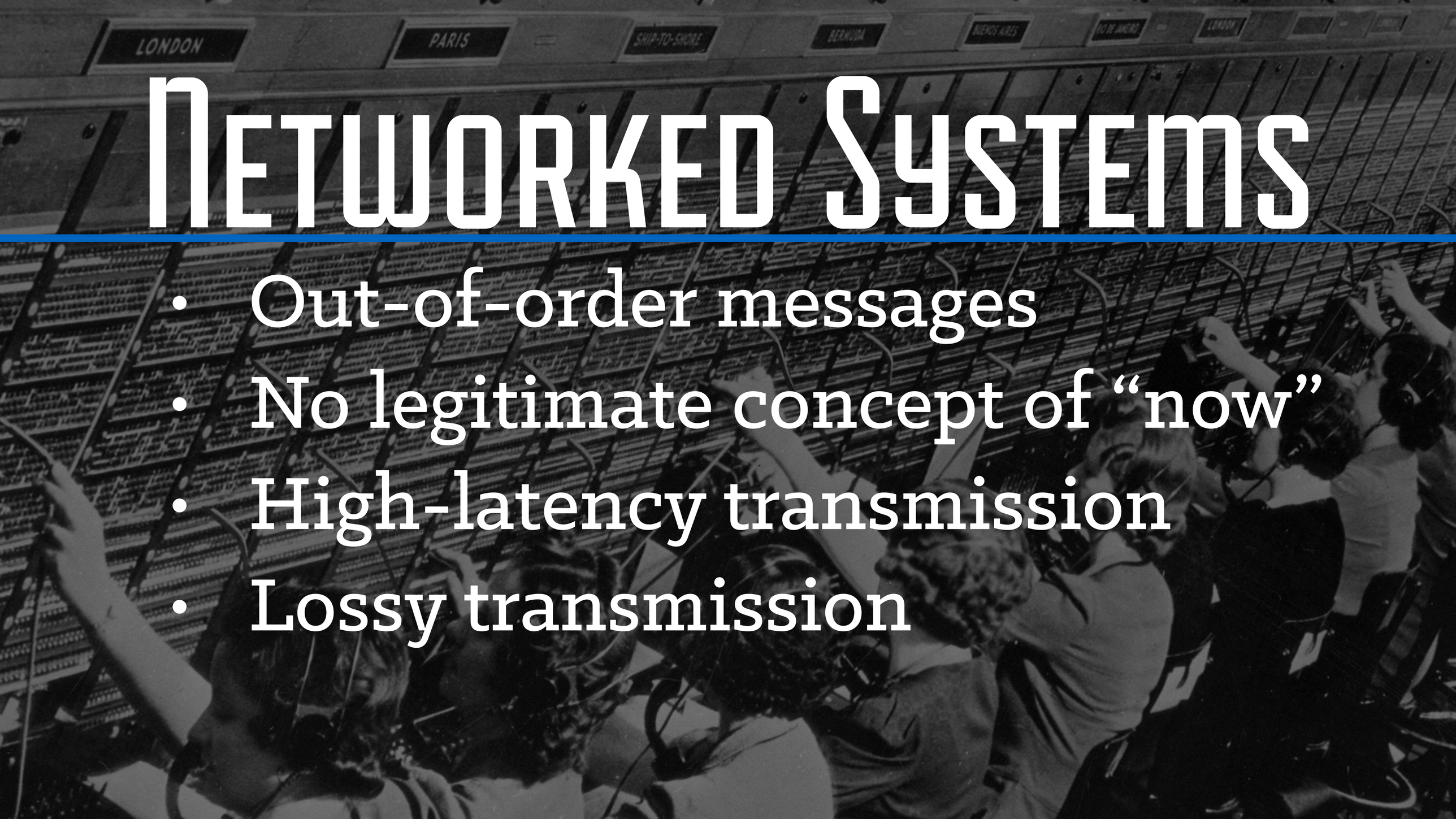


Networked Systems

- Out-of-order messages
- No legitimate concept of “now”

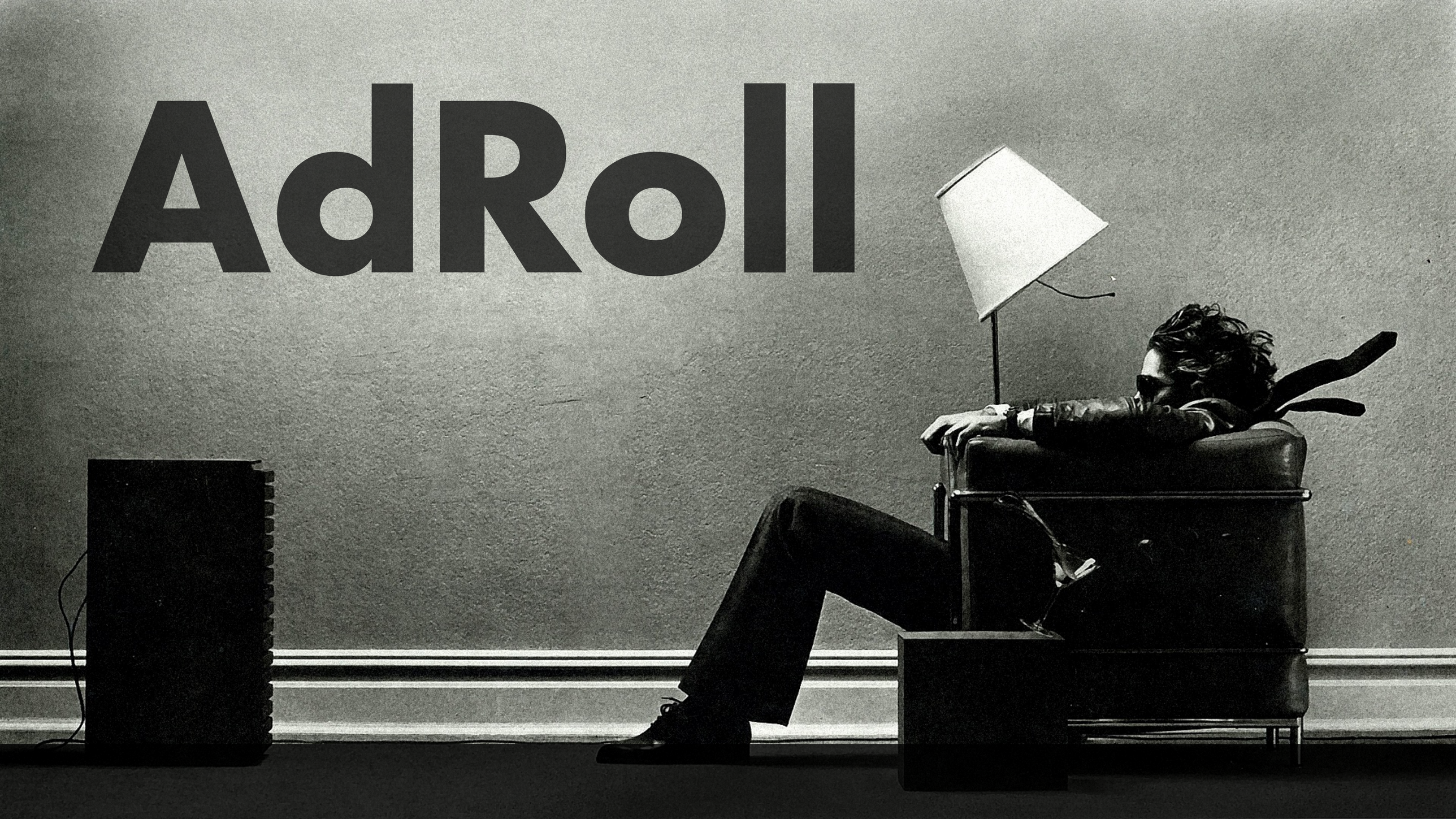
Networked Systems

- Out-of-order messages
- No legitimate concept of “now”
- High-latency transmission



Networked Systems

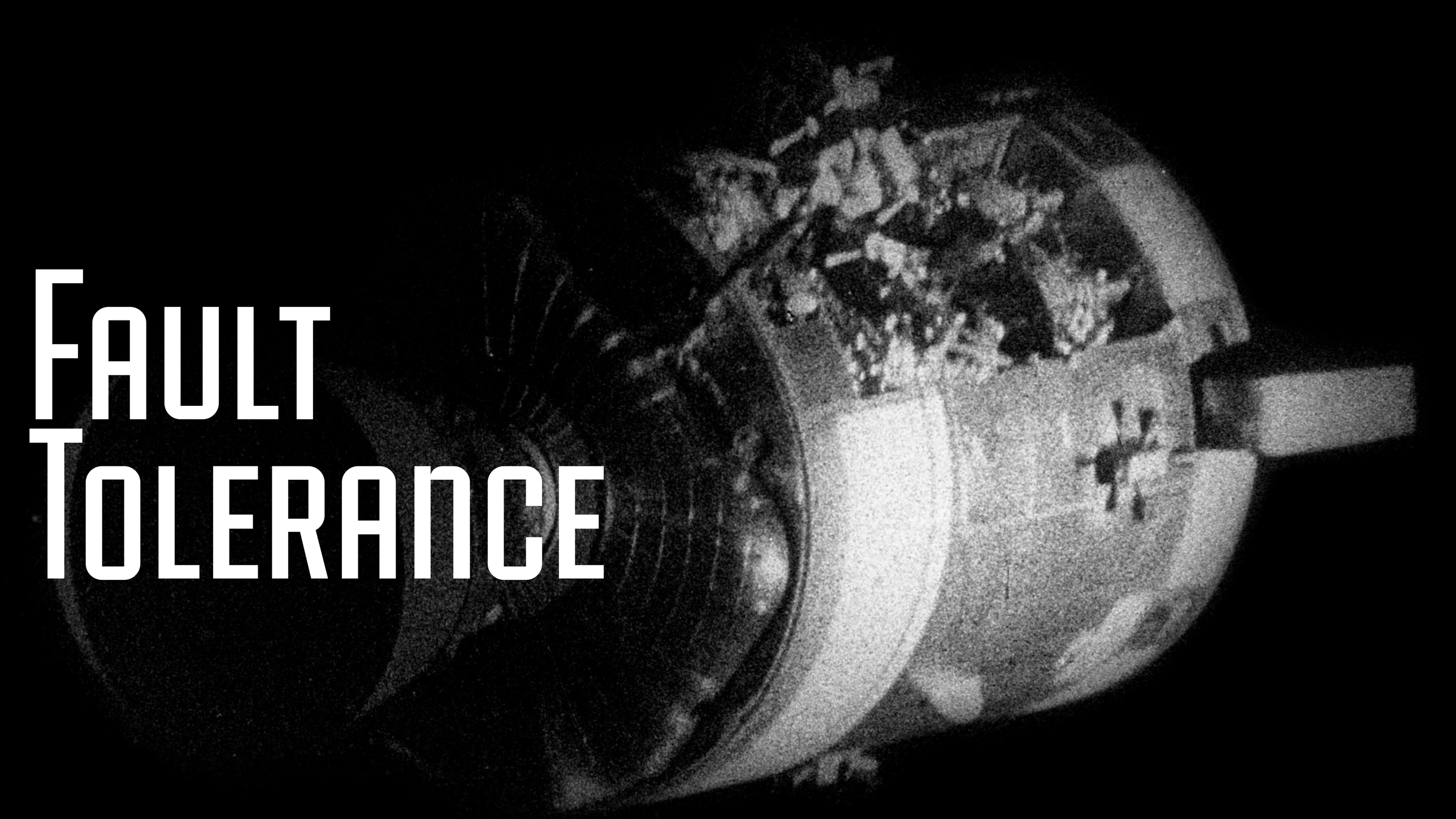
- Out-of-order messages
- No legitimate concept of “now”
- High-latency transmission
- Lossy transmission

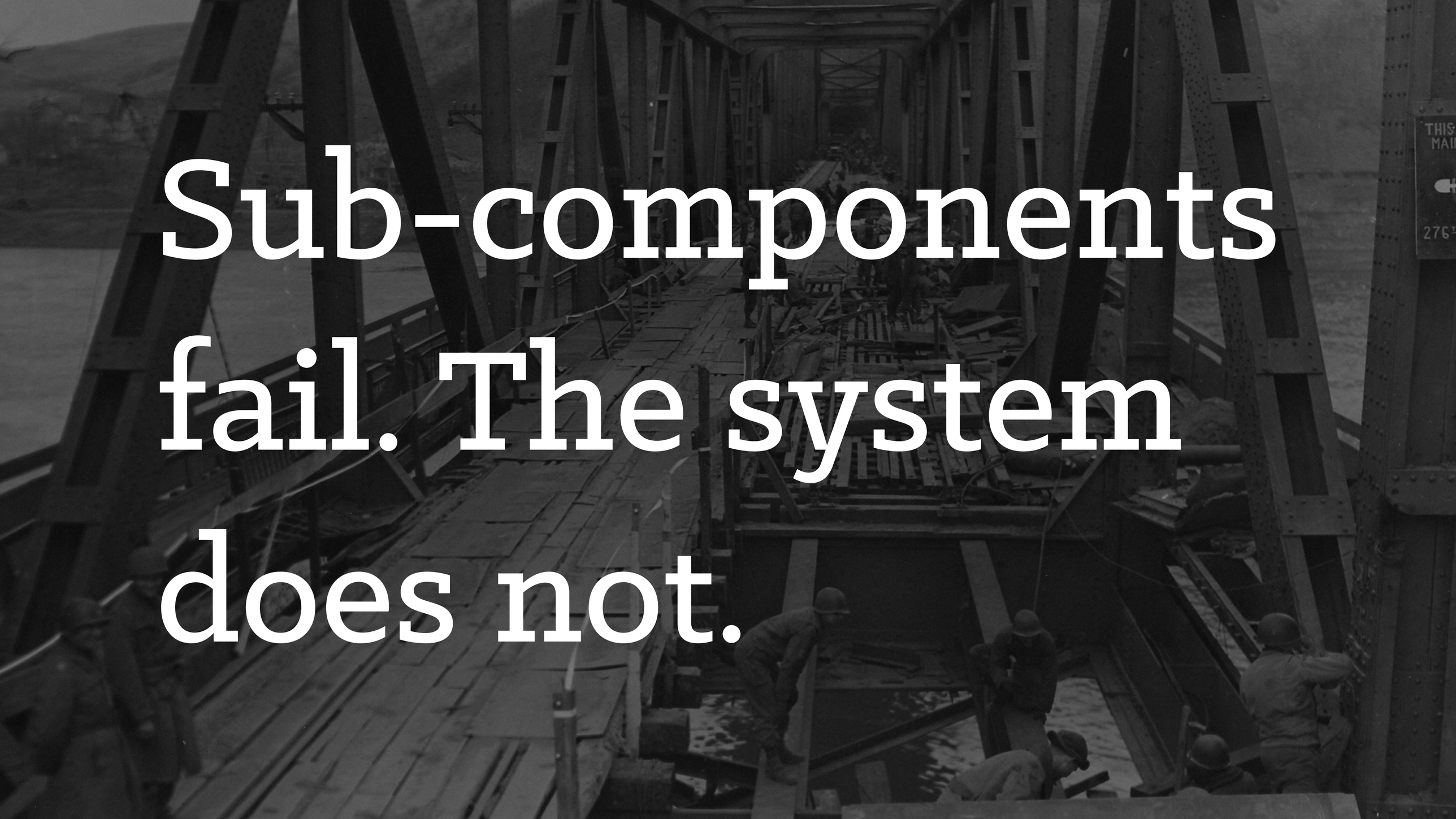

AdRoll


REAL-TIME
BIDDING


Erlang

Fault Tolerance





Sub-components fail. The system does not.


Well, not right
away...


What’s it take?

Option 1: Perfection

Option 1: Perfection

- Total control over the whole mechanism.

Option 1: Perfection

- Total control over the whole mechanism.
- Total understanding of the problem domain.

Option 1: Perfection

- Total control over the whole mechanism.
- Total understanding of the problem domain.
- Specific, explicit system goals.

Option 1: Perfection

- Total control over the whole mechanism.
- Total understanding of the problem domain.
- Specific, explicit system goals.
- Well-known service lifetime.

Option 1: Perfection

“They Write the Right Stuff”
Fast Company, 2005

Option 1: Perfection

- Extremely expensive.

Option 1: Perfection

- Extremely expensive.
- Intentionally stifles creativity.

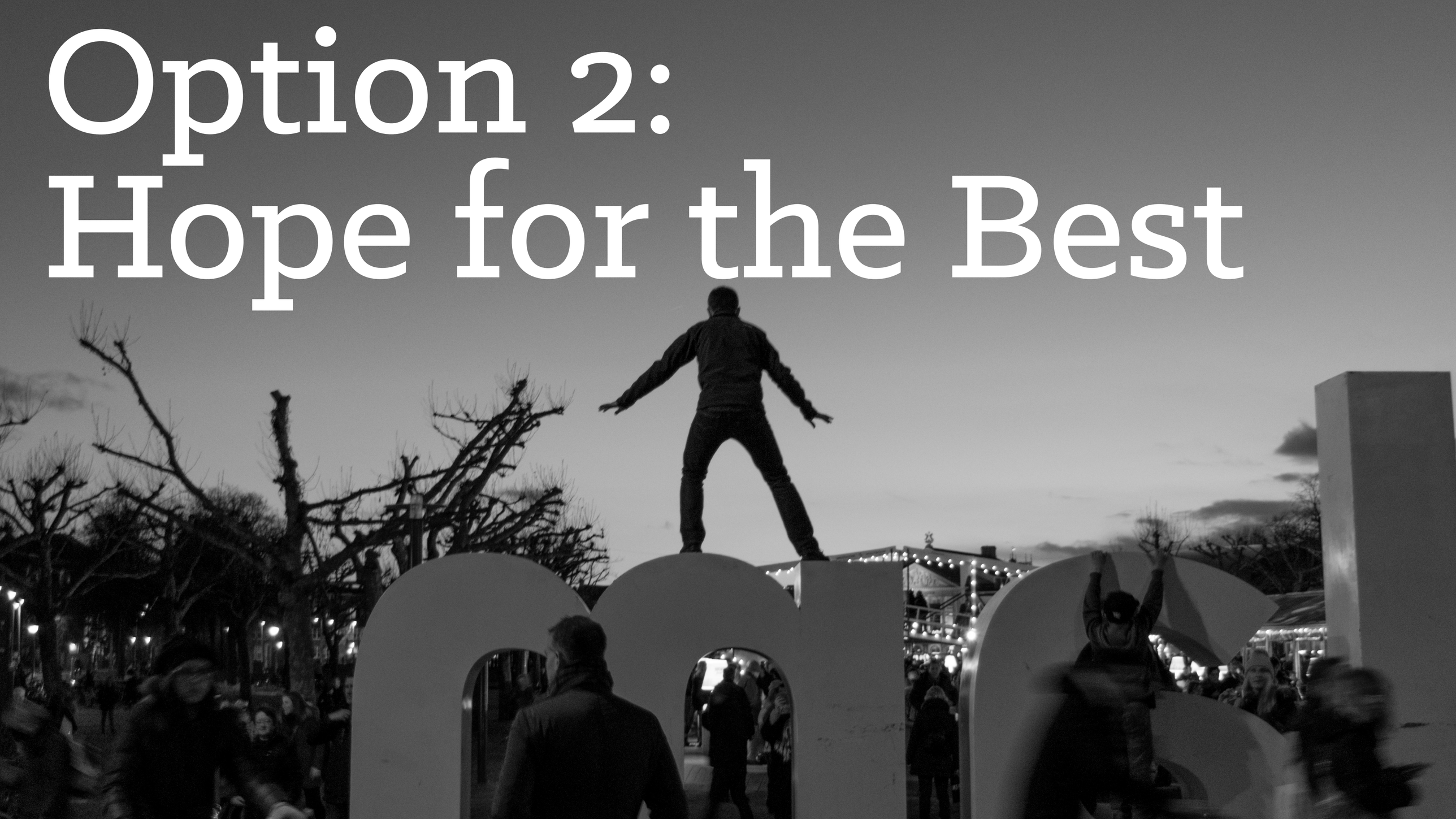
Option 1: Perfection

- Extremely expensive.
- Intentionally stifles creativity.
- Design up front.

Option 1: Perfection

- Extremely expensive.
- Intentionally stifles creativity.
- Design up front.
- Complete control of the system is not complete.

Option 2:
Hope for the Best



Option 2: Hope for the Best

- Little up-front knowledge of the problem domain.

Option 2: Hope for the Best

- Little up-front knowledge of the problem domain.
- Implicit or short-term system goals.

Option 2: Hope for the Best

- Little up-front knowledge of the problem domain.
- Implicit or short-term system goals.
- No money down.

Option 2: Hope for the Best

- Little up-front knowledge of the problem domain.
- Implicit or short-term system goals.
- No money down.
- Inginutity under pressure.

Option 2: Hope for the Best

"Move fast and break things!"

Option 2: Hope for the Best

- Ignorance of problem domain leads to long-term system issues.

Option 2: Hope for the Best

- Ignorance of problem domain leads to long-term system issues.
- Failures *do* propigate out toward users.

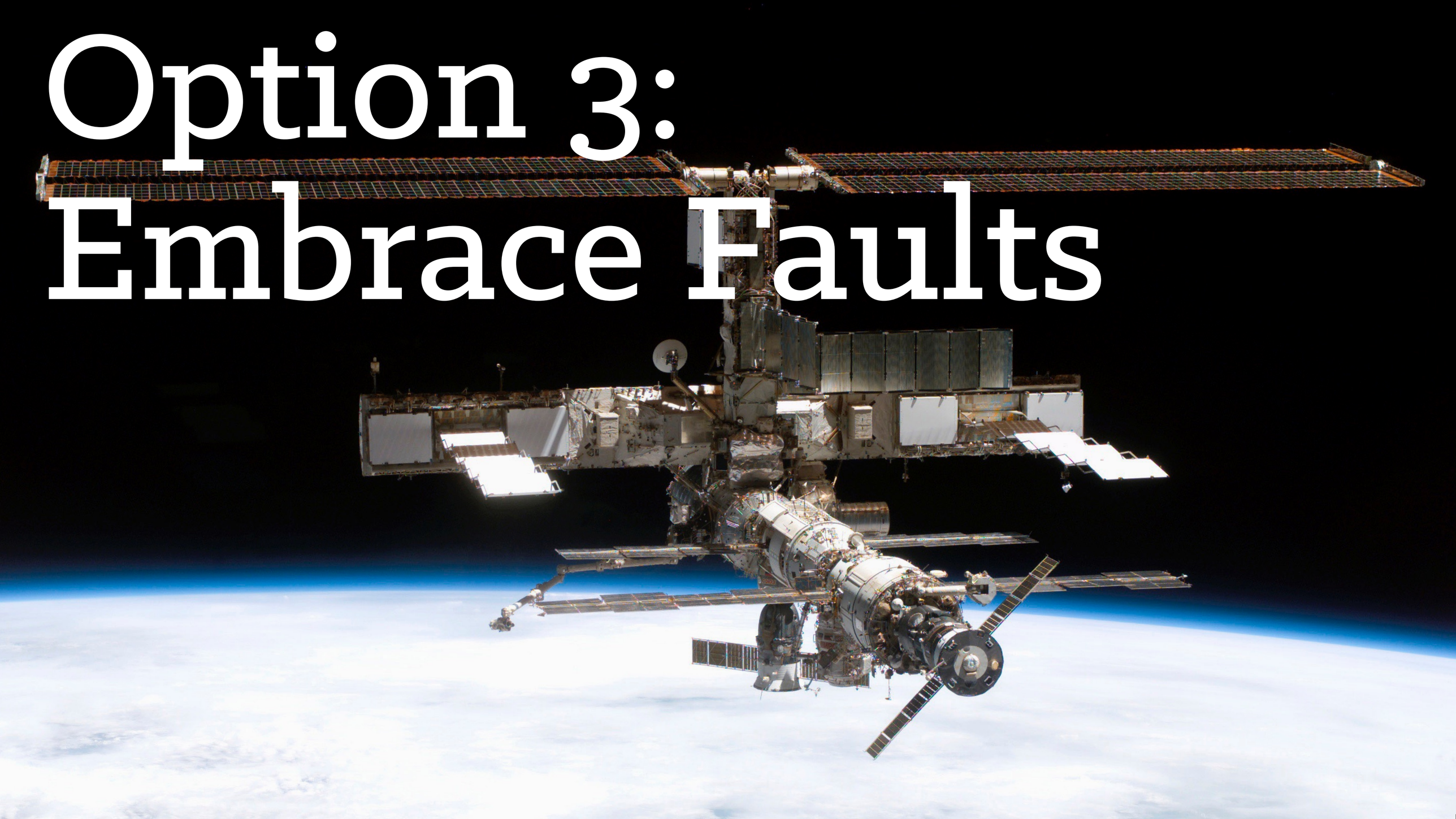
Option 2: Hope for the Best

- Ignorance of problem domain leads to long-term system issues.
- Failures *do* propigate out toward users.
- No, money down!

Option 2: Hope for the Best

- Ignorance of problem domain leads to long-term system issues.
- Failures *do* propigate out toward users.
- No, money down!
- Hard to change cultural values.

Option 3: Embrace Faults



Option 3: Embrace Faults

- Partial control over the whole mechanism.

Option 3: Embrace Faults

- Partial control over the whole mechanism.
- Partial understanding of the problem domain.

Option 3: Embrace Faults

- Partial control over the whole mechanism.
- Partial understanding of the problem domain.
- Sorta explicit system goals.

Option 3: Embrace Faults

- Partial control over the whole mechanism.
- Partial understanding of the problem domain.
- Sorta explicit system goals.
- Able to spot a failure when you see one.

Option 3: Embrace Faults

"Fail fast. Either do the right thing or stop."

"Why Do Computers Stop and What Can Be Done About it?", Jim Gray, 1985 (paraphrase)

Option 3: Embrace Faults

- Faults are isolated but *must* be resolved in production.

Option 3: Embrace Faults

- Faults are isolated but *must* be resolved in production.
- Must carefully design for introspection.

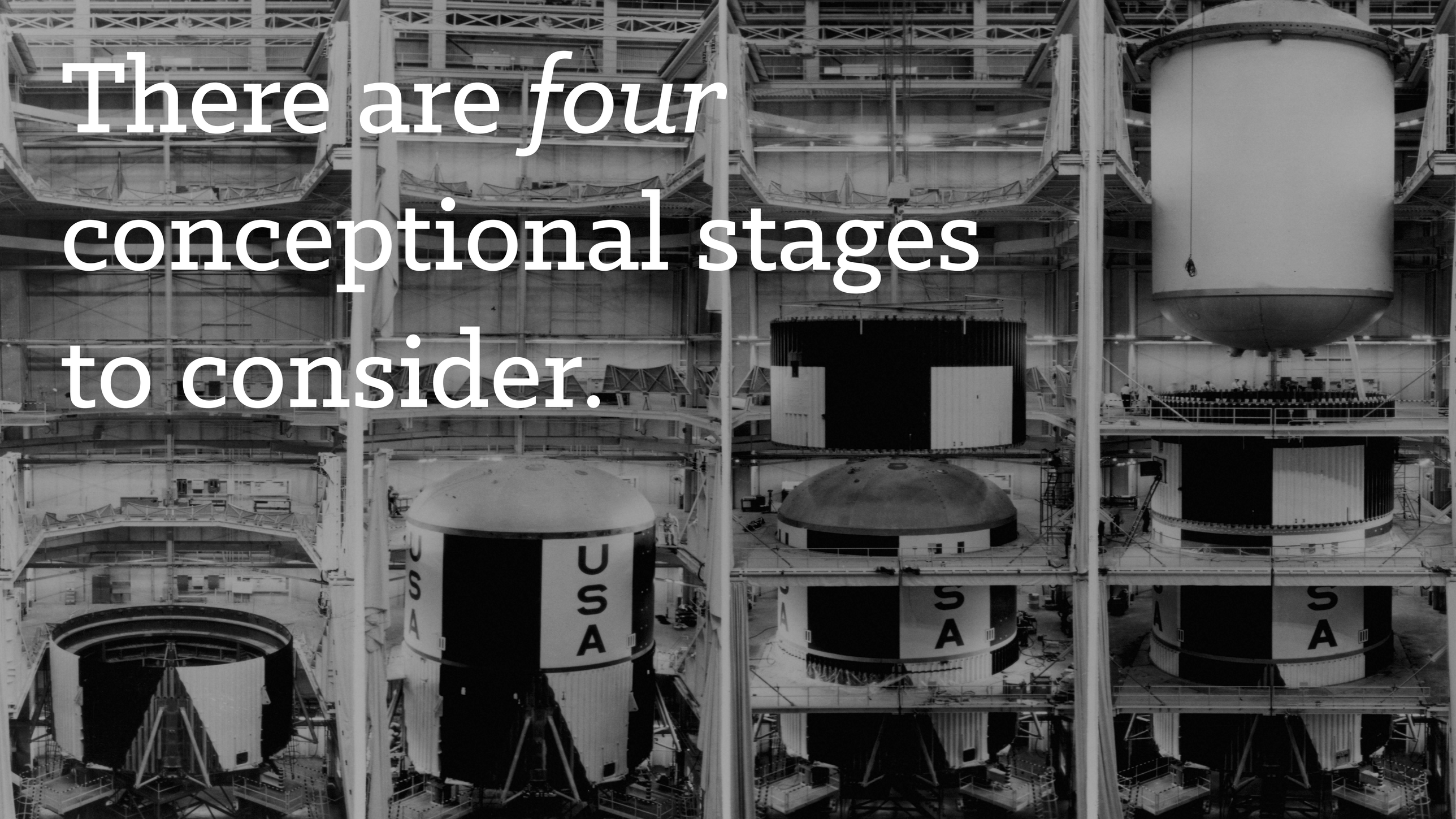
Option 3: Embrace Faults

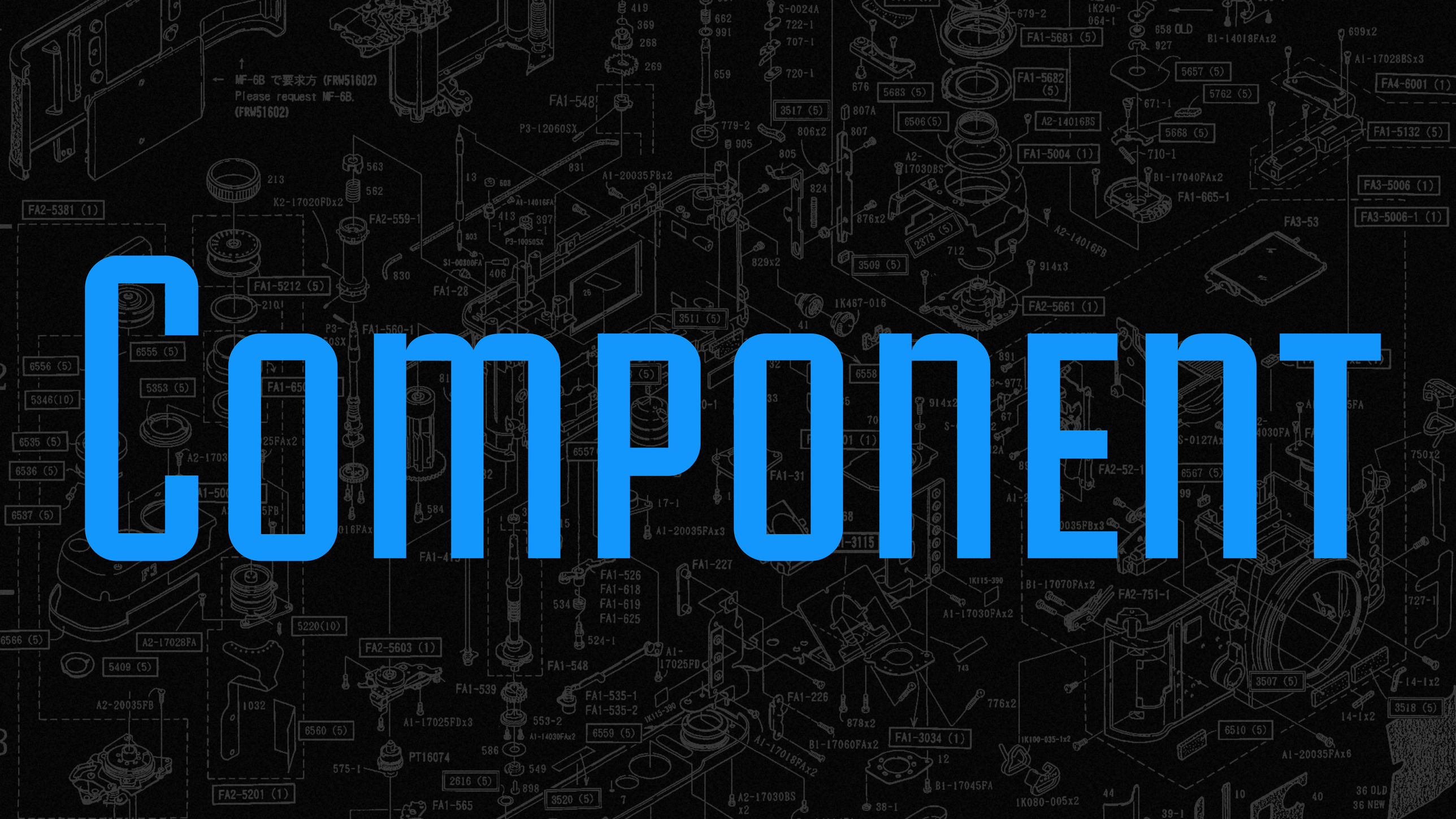
- Faults are isolated but *must* be resolved in production.
- Must carefully design for introspection.
- Moderate design up-front.

Option 3: Embrace Faults

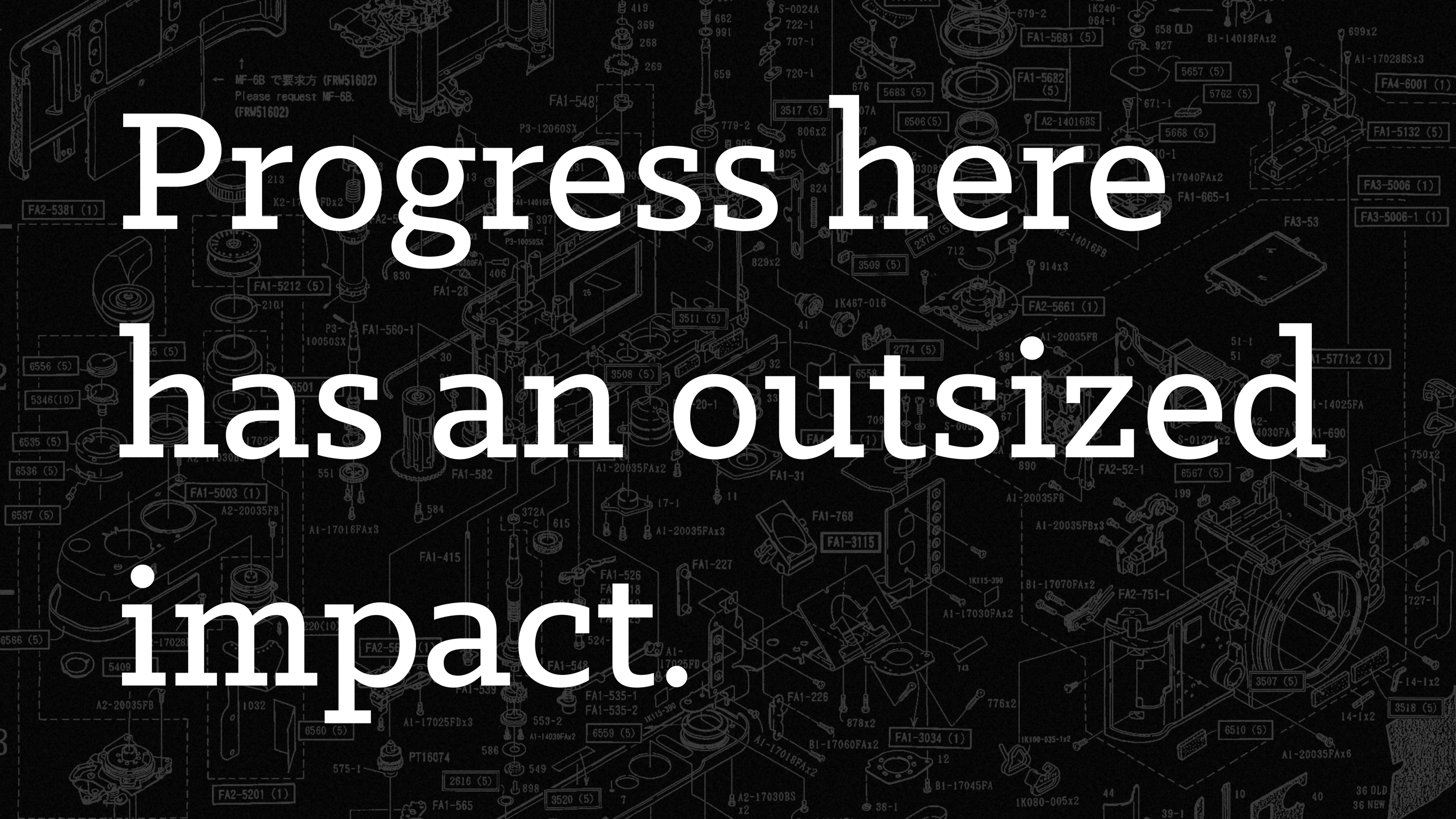
- Faults are isolated but *must* be resolved in production.
- Must carefully design for introspection.
- Moderate design up-front.
- Pay a little now, pay a little later.

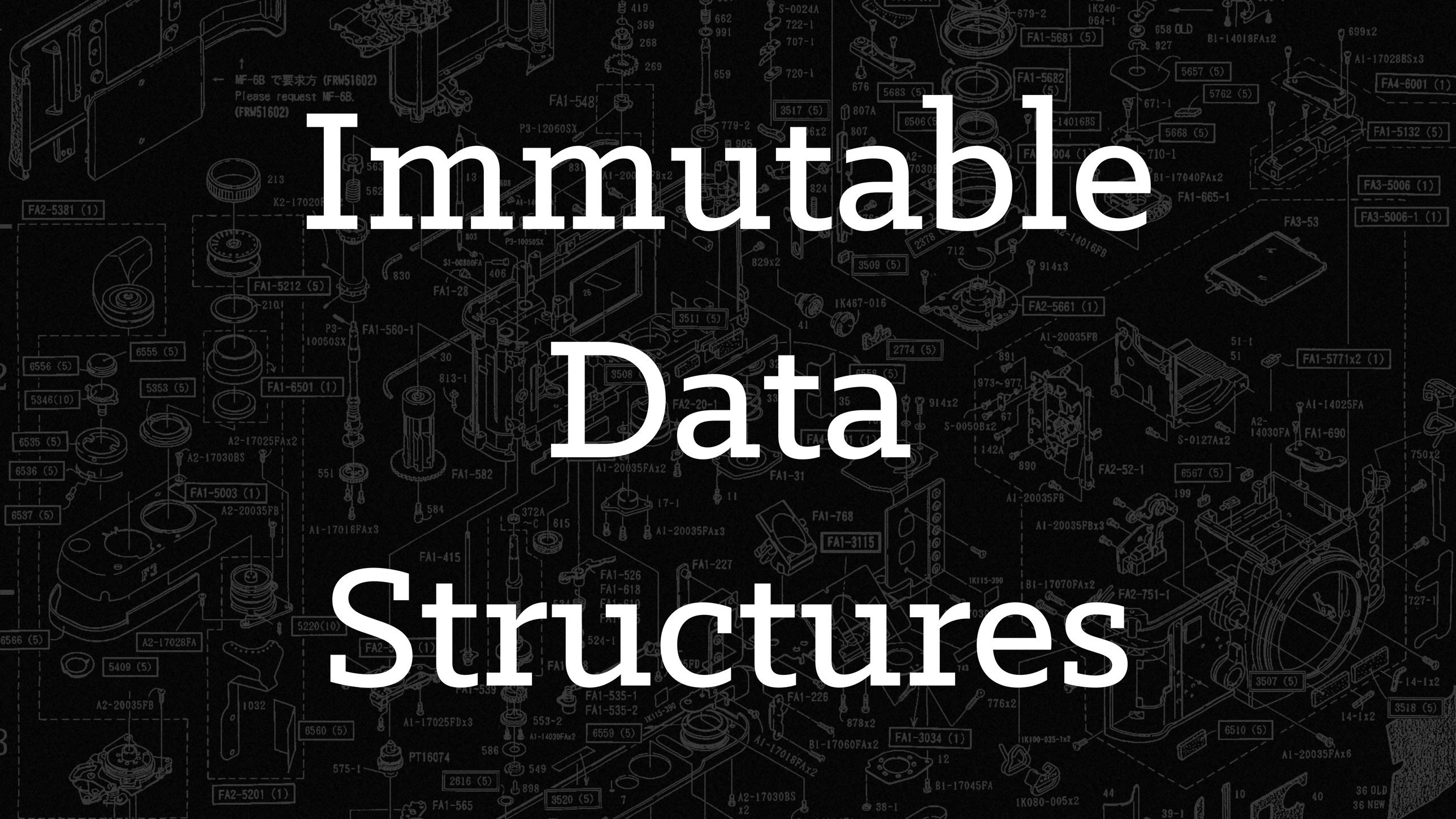

Let’s talk
embracing
faults.


There are *four*
conceptional stages
to consider.

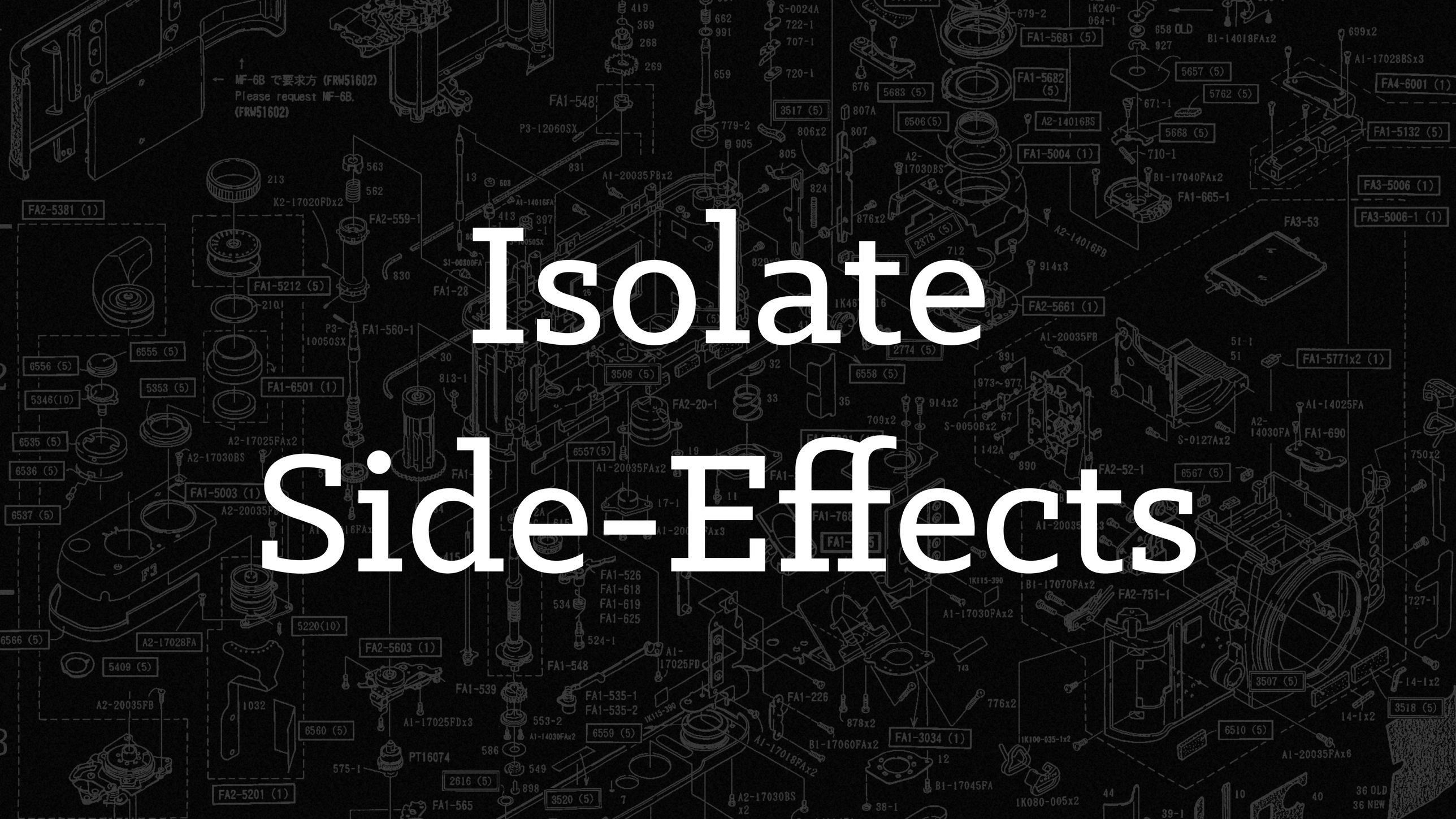

COMPONENT


The most
atomic level
of the system.

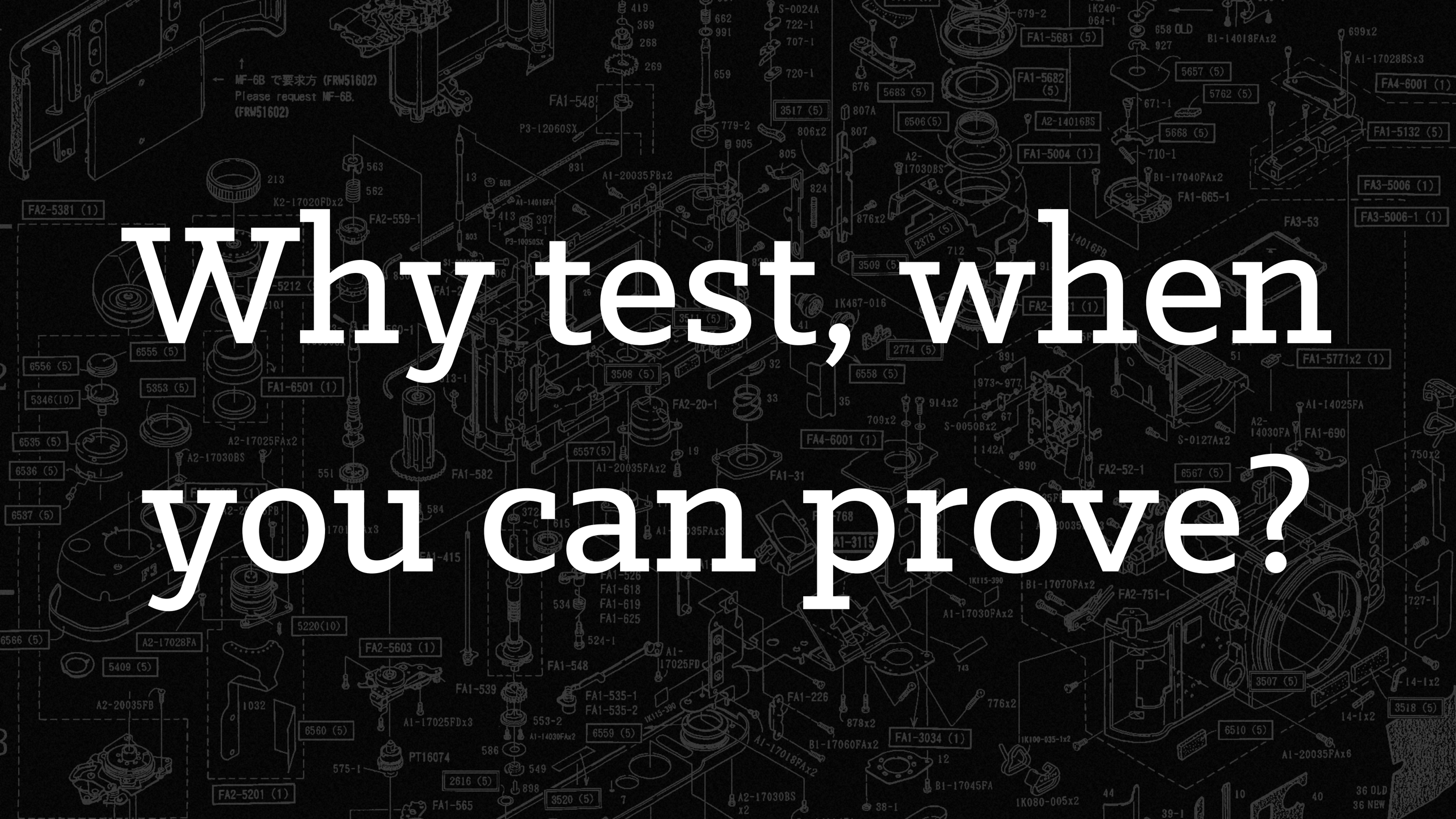

Progress here
has an outsized
impact.

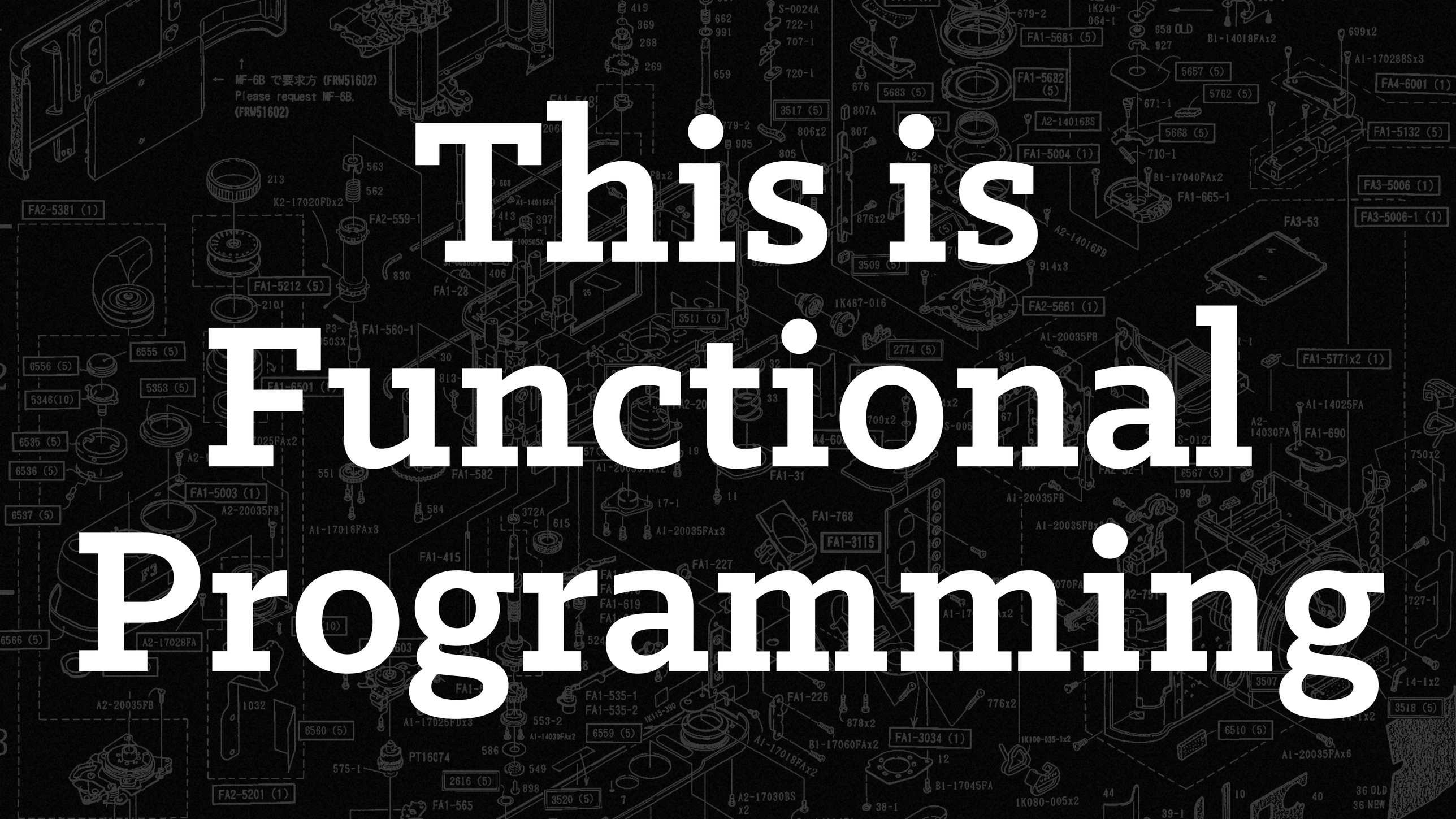


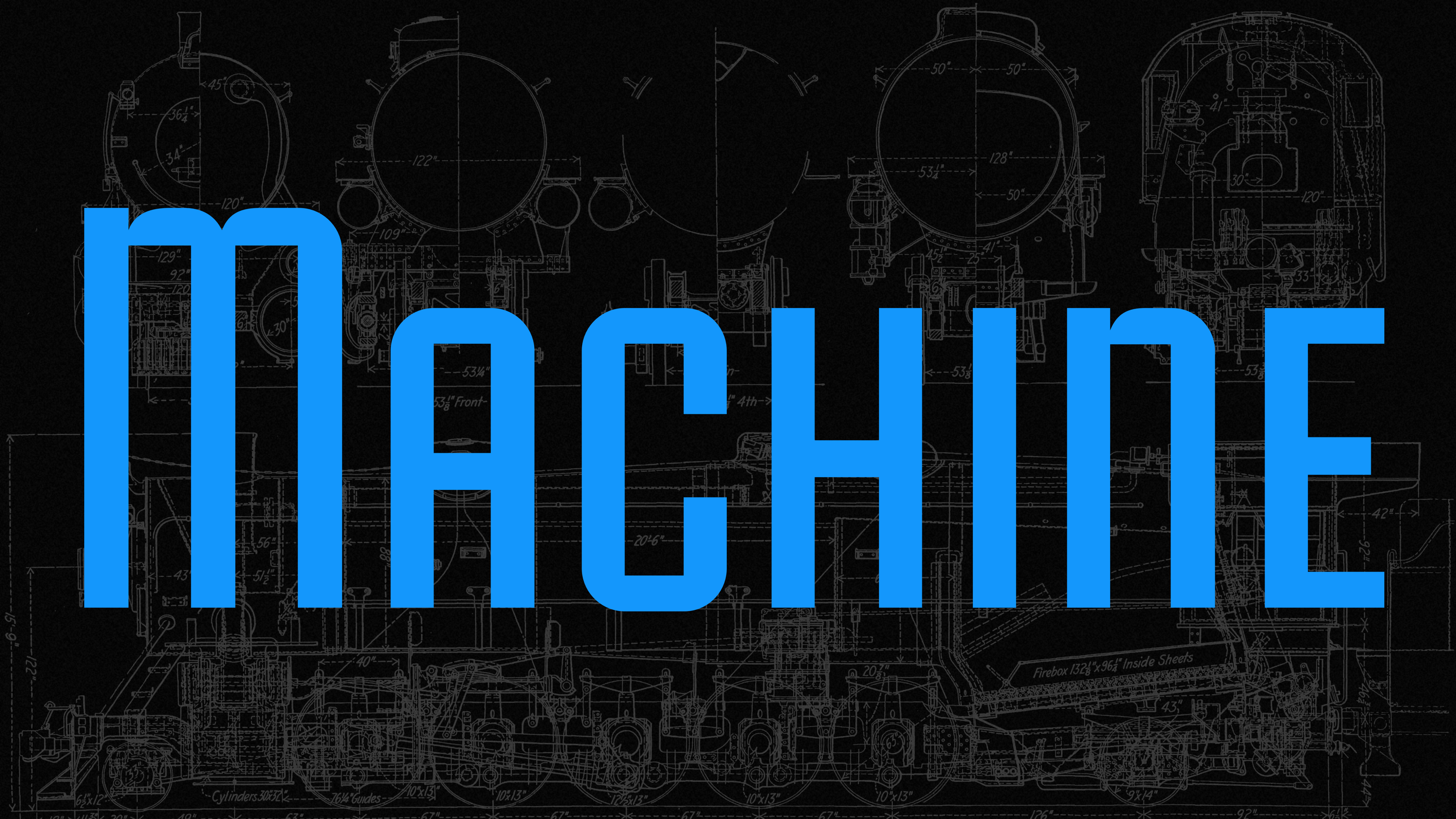
Immutable Data Structures

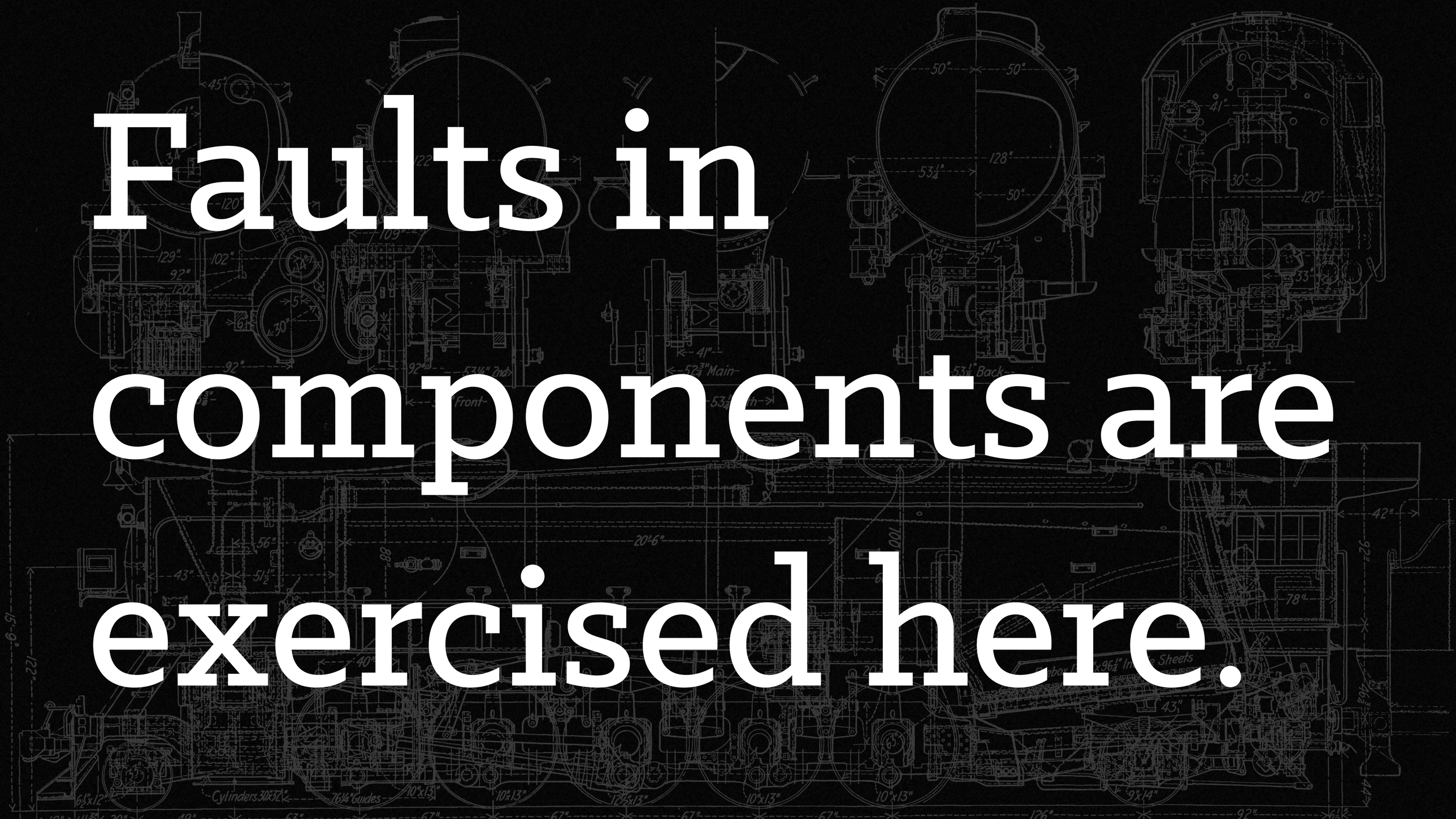

Isolate
Side-Effects


Compile-Time
Guarantees

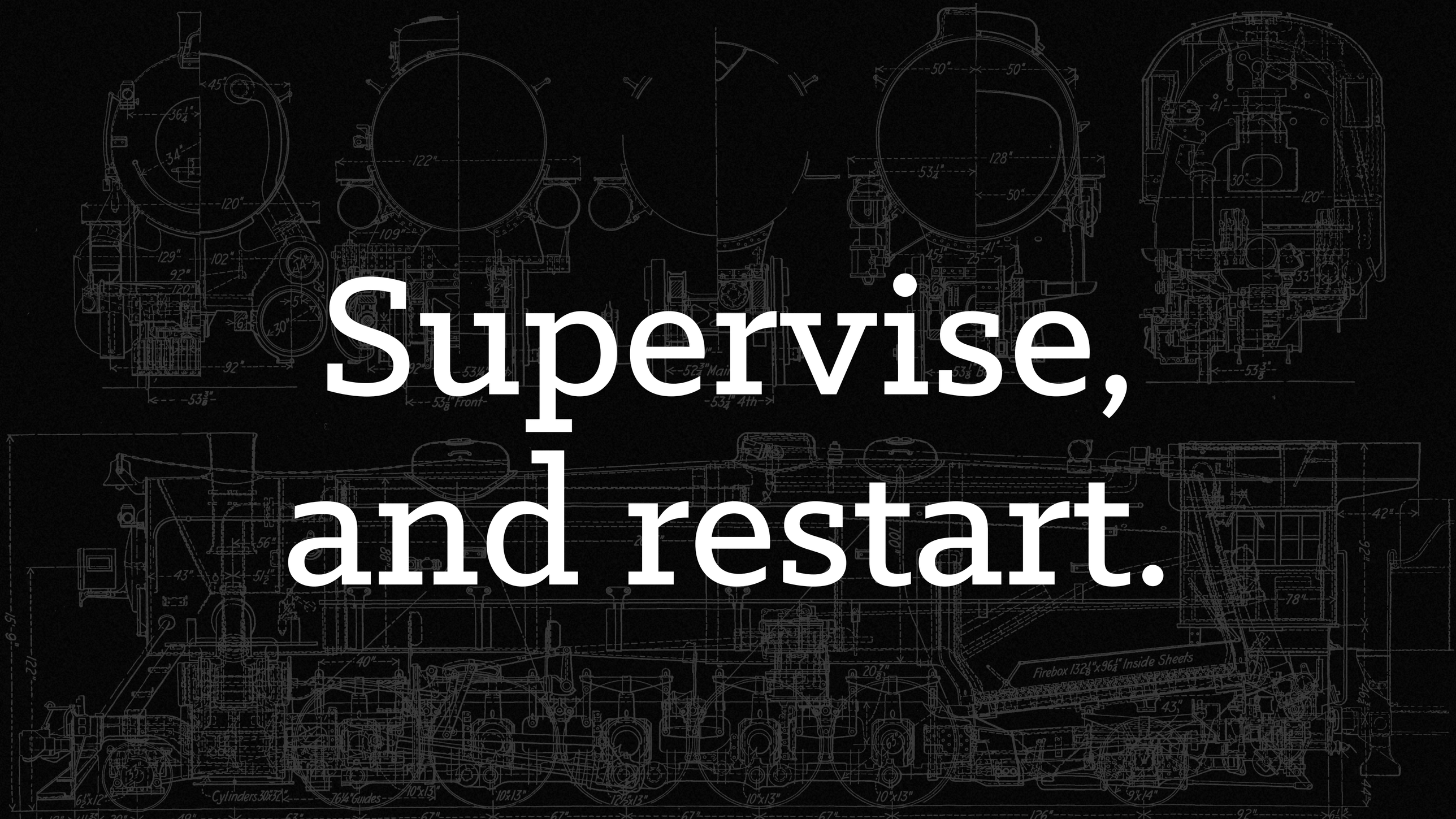

Why test, when
you can prove?

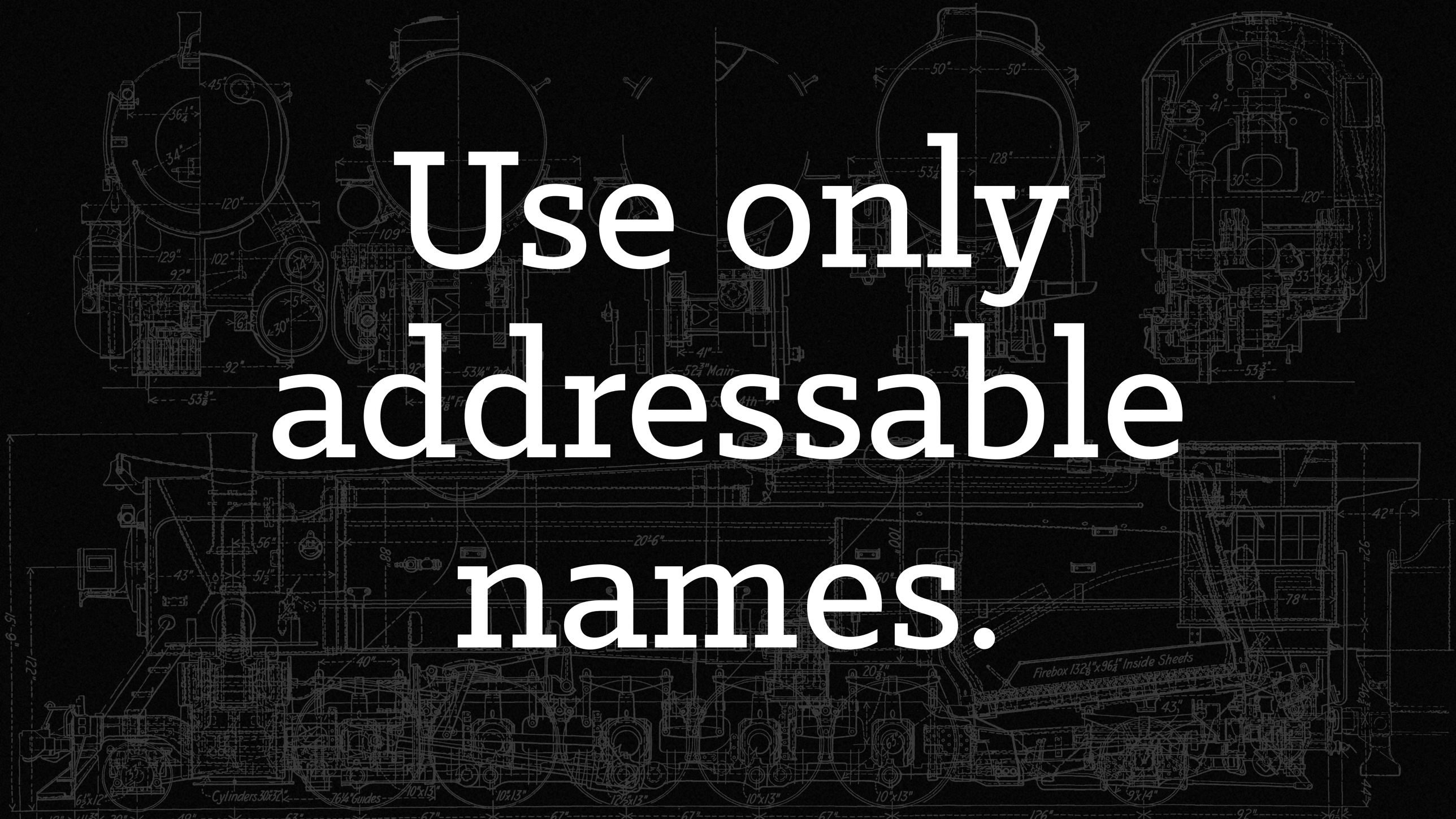

This is
Functional
Programming

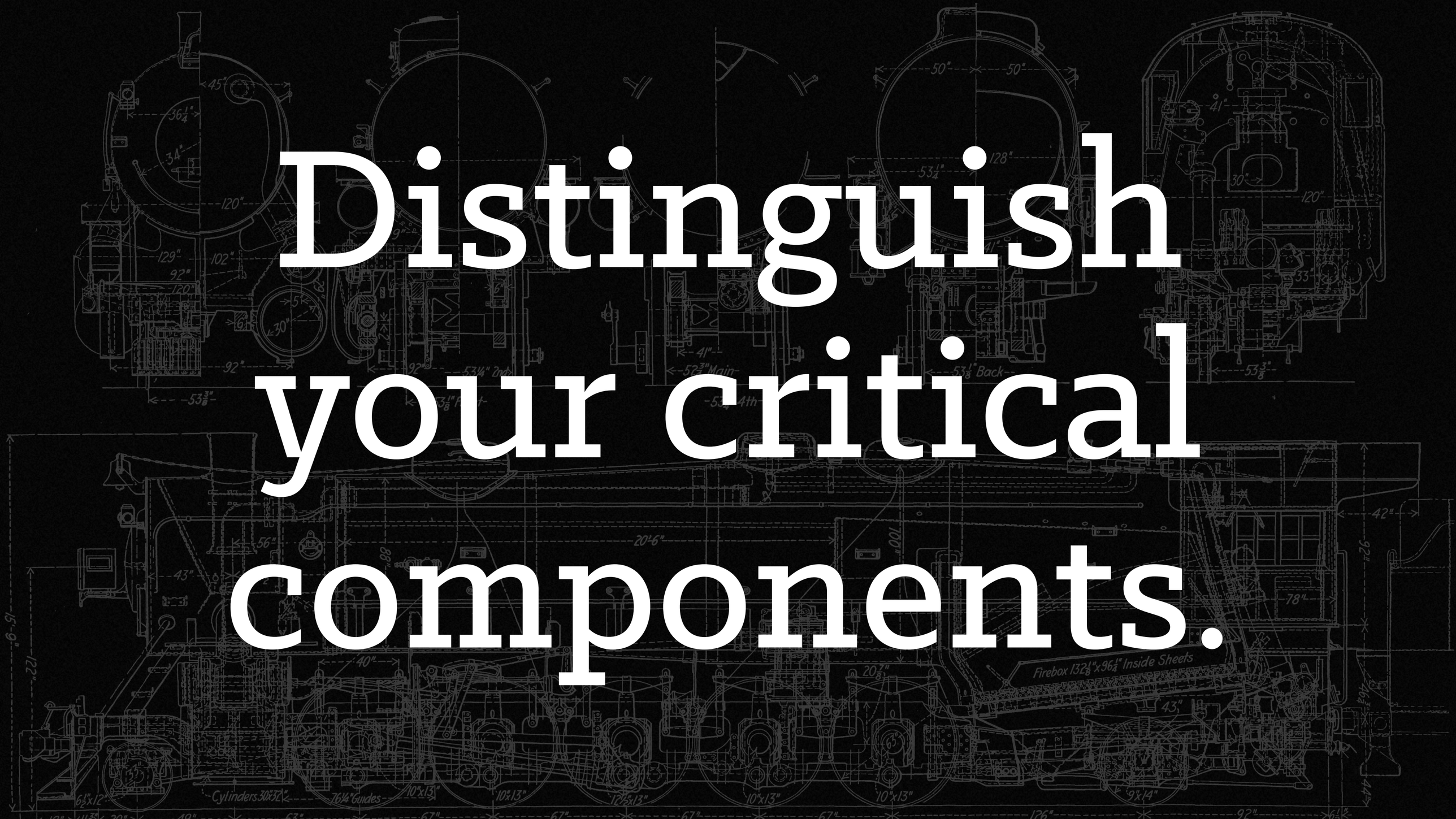

MACHINE

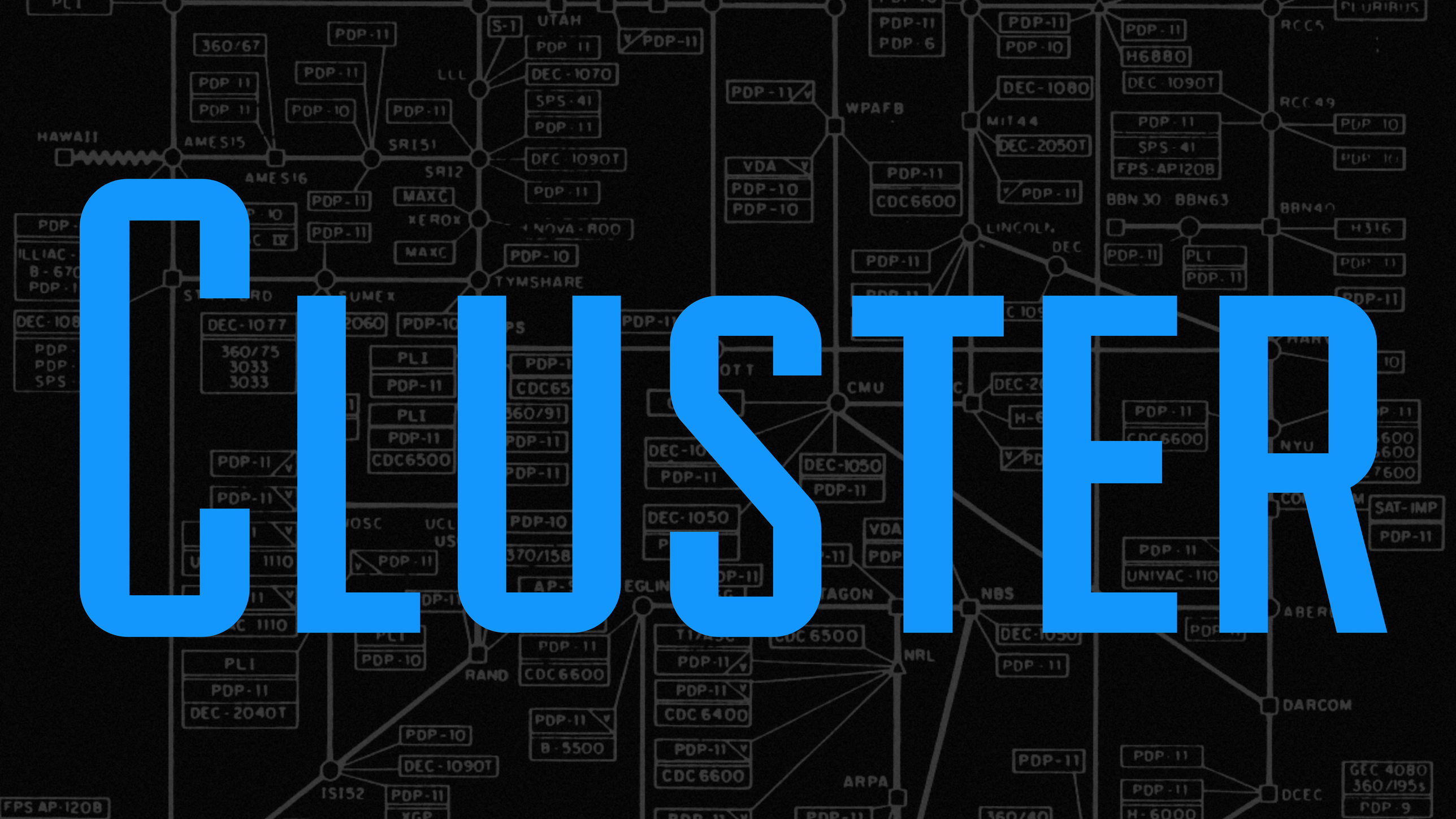

Faults in components are exercised here.

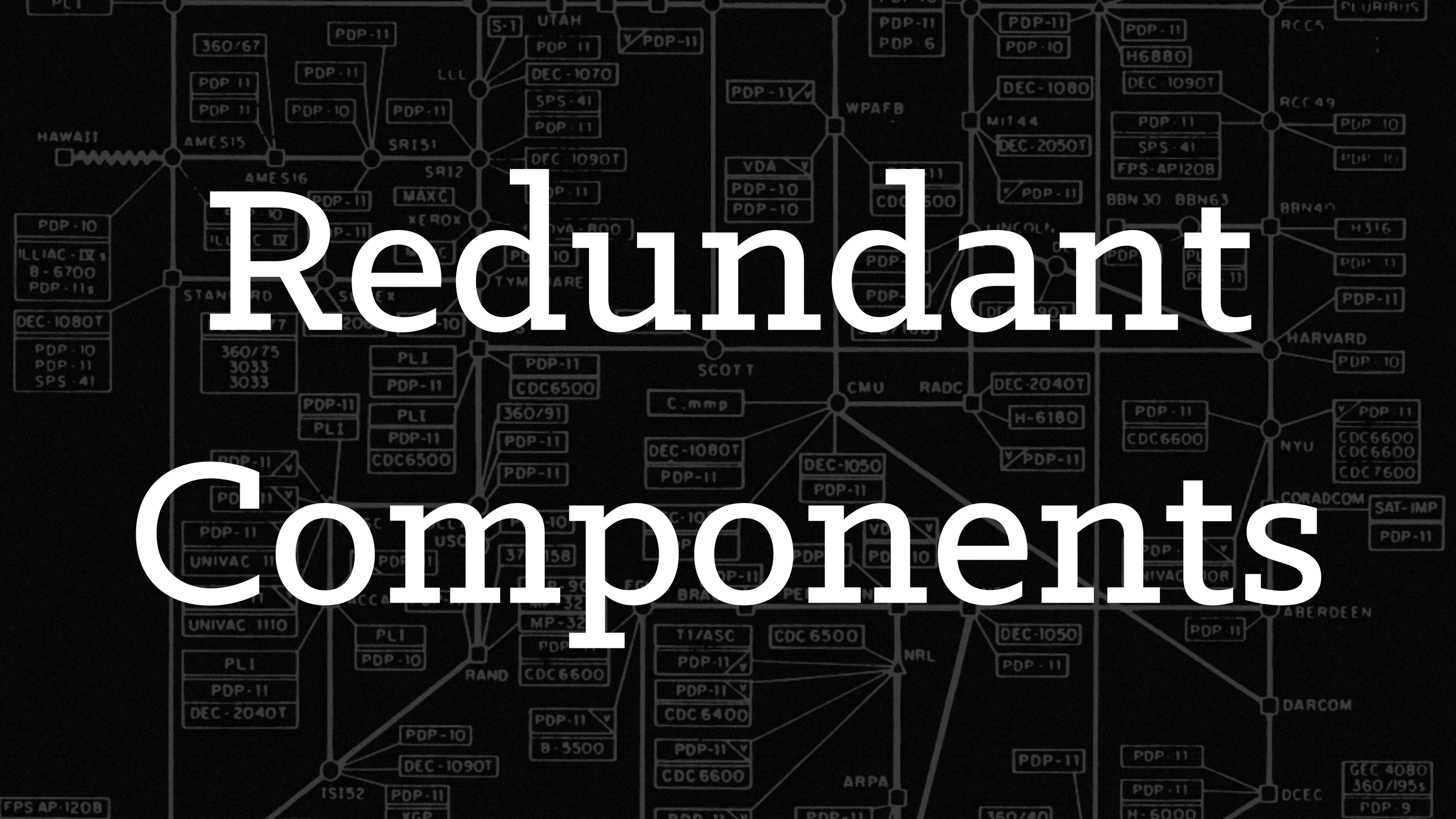

Faults in
interactions are
exercised here.

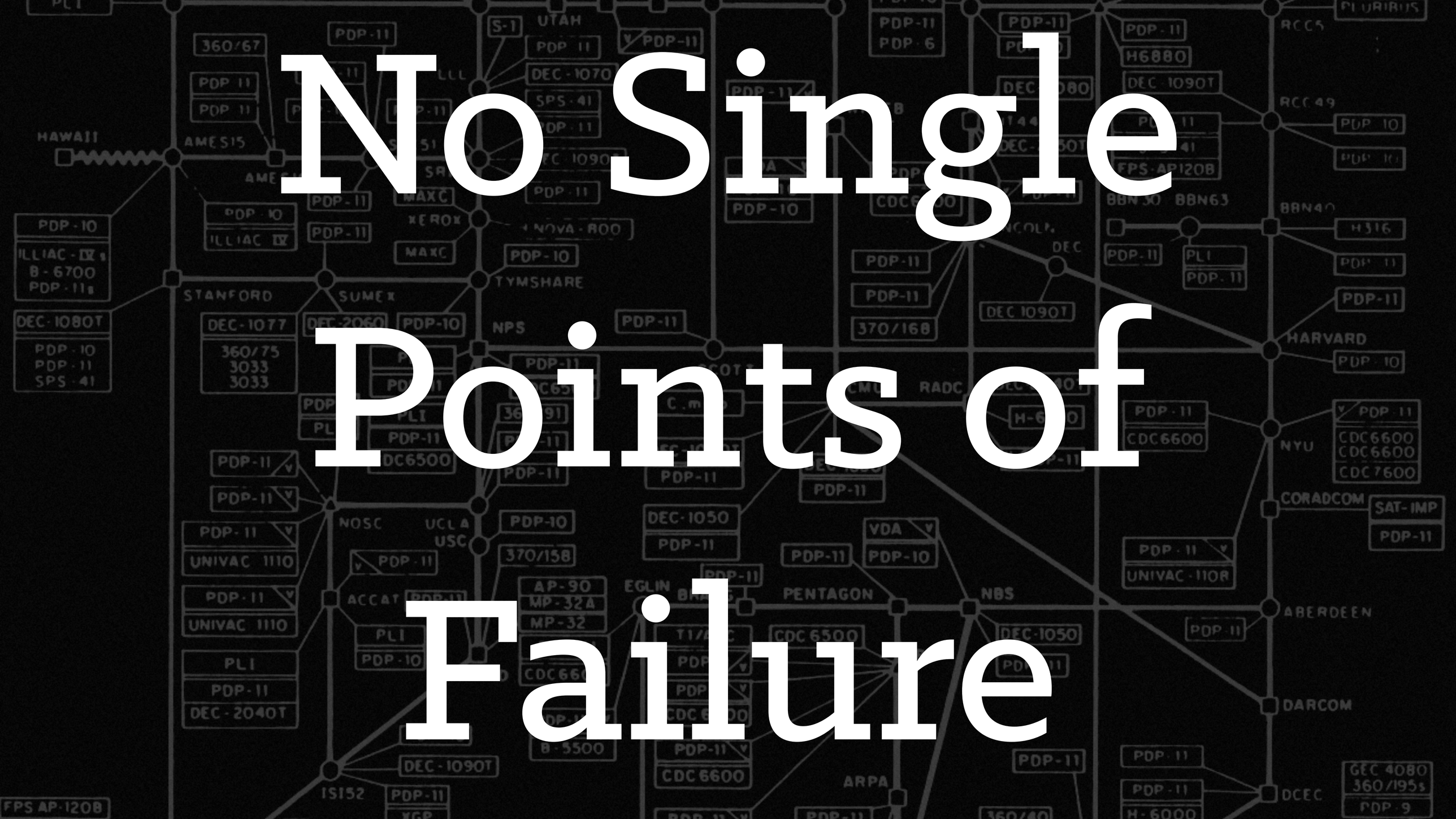

Supervise,
and restart.


Use only
addressable
names.

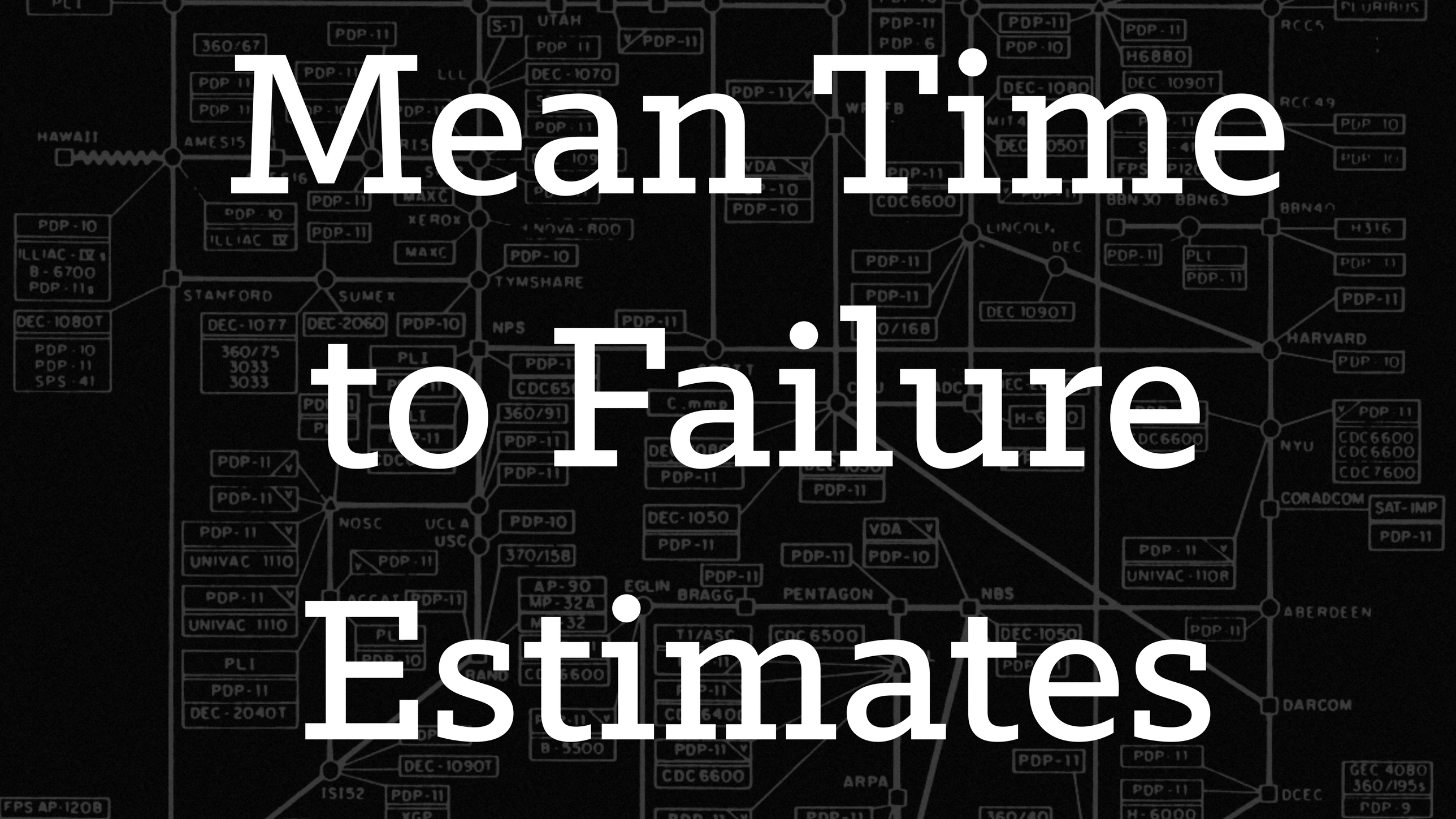

Distinguish
your critical
components.

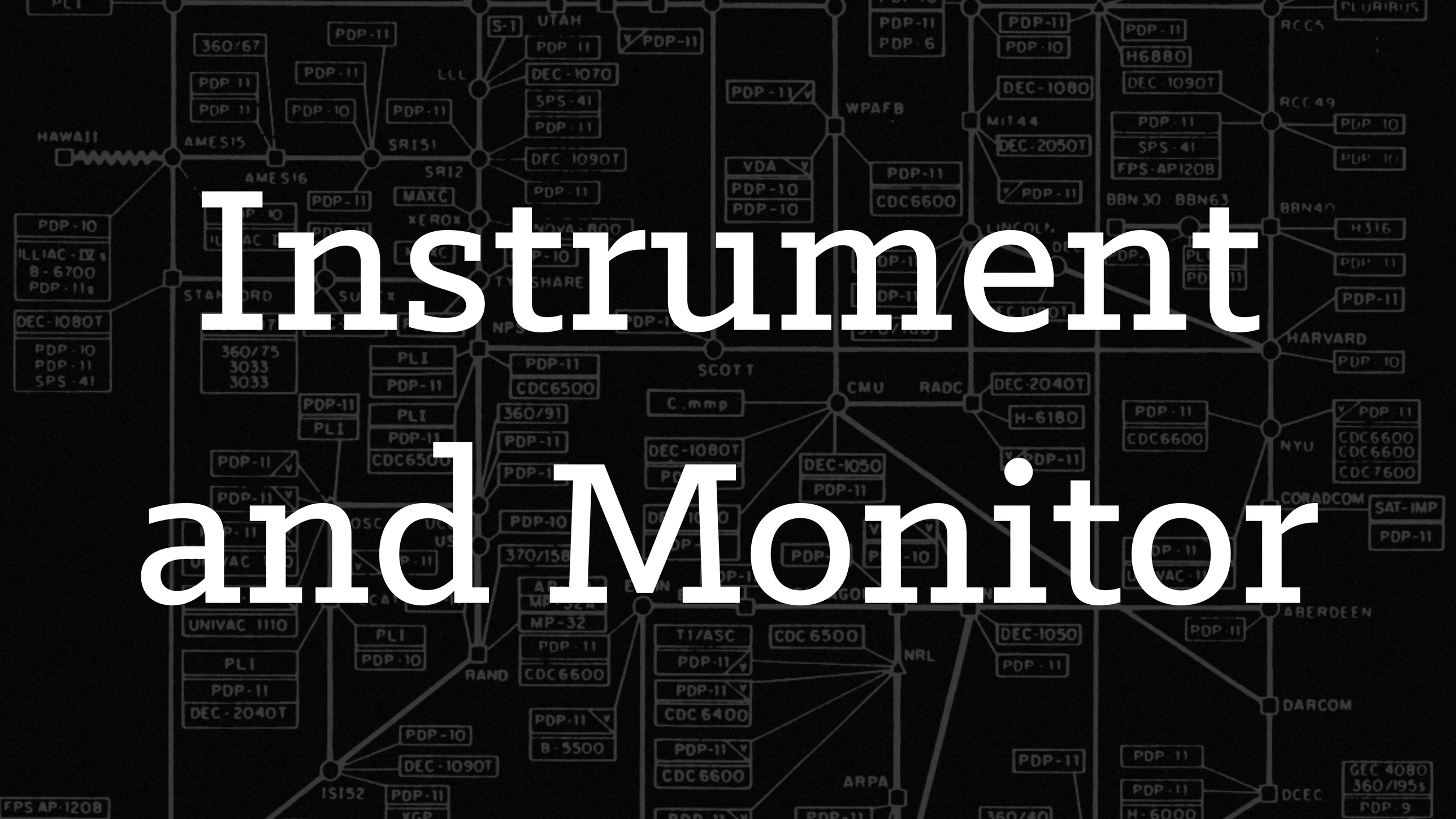

CLUSTER
UTAH
LLL
HAWAII
AMES15
AMES16
SRI51
SRI2
MAXC
XEROX
TYMSHARE
WPAFB
CMU
LINCOLN
DEC
NYU
RAND
NRL
NBS
ARPA
DARCOM
DCEC
ISI52
PDP-11
PDP-10
DEC-1090T
DEC-1070
DEC-1080
DEC-2050T
CDC6600
CDC6500
CDC6400
SPS-41
PLI
UNIVAC-1100
FPS AP-120B
SAT-IMP
PLURIBUS


Redundant
Components



No Single Points of Failure


Mean Time
to Failure
Estimates


Instrument
and Monitor
UTAH
LLL
WPAFB
HAWAII
AMES15
AMES16
SRI51
SRI2
MAXC
XEROX
SCOTT
CMU
RADC
RAND
NRL
ARPA
HARVARD
NYU
CORADCOM
ABERDEEN
DARCOM
DCEC
BBN30
BBN63
BBN40
RCC5
RCC49
PLURIBUS
MIT44
LINCOLN
ISI52
PDP-11
PDP-10
DEC-1090T
DEC-1080
DEC-1080T
DEC-2050T
DEC-2040T
DEC-1050
CDC6500
CDC6600
CDC6400
CDC 6500
CDC 6600
CDC 6400
CDC6600
CDC6600
CDC7600
C.mmp
T1/ASC
MP-32
B-5500
ILLIAC-IV
B-6700
PDP-11s
SPS-41
FPS-AP120B
H6180
H6880
H316
UNIVAC 1110
360/67
360/75
3033
3033
360/91
370/158
PLI
SAT-IMP
GEC 4080
360/195s
PDP-9


ORGANIZATION

A finely built machine *without* a supporting organization is a disaster waiting to happen.

Chernobyl

STS-51-L

Deepwater Horizon

Magnitogorsk

Damascus Incident

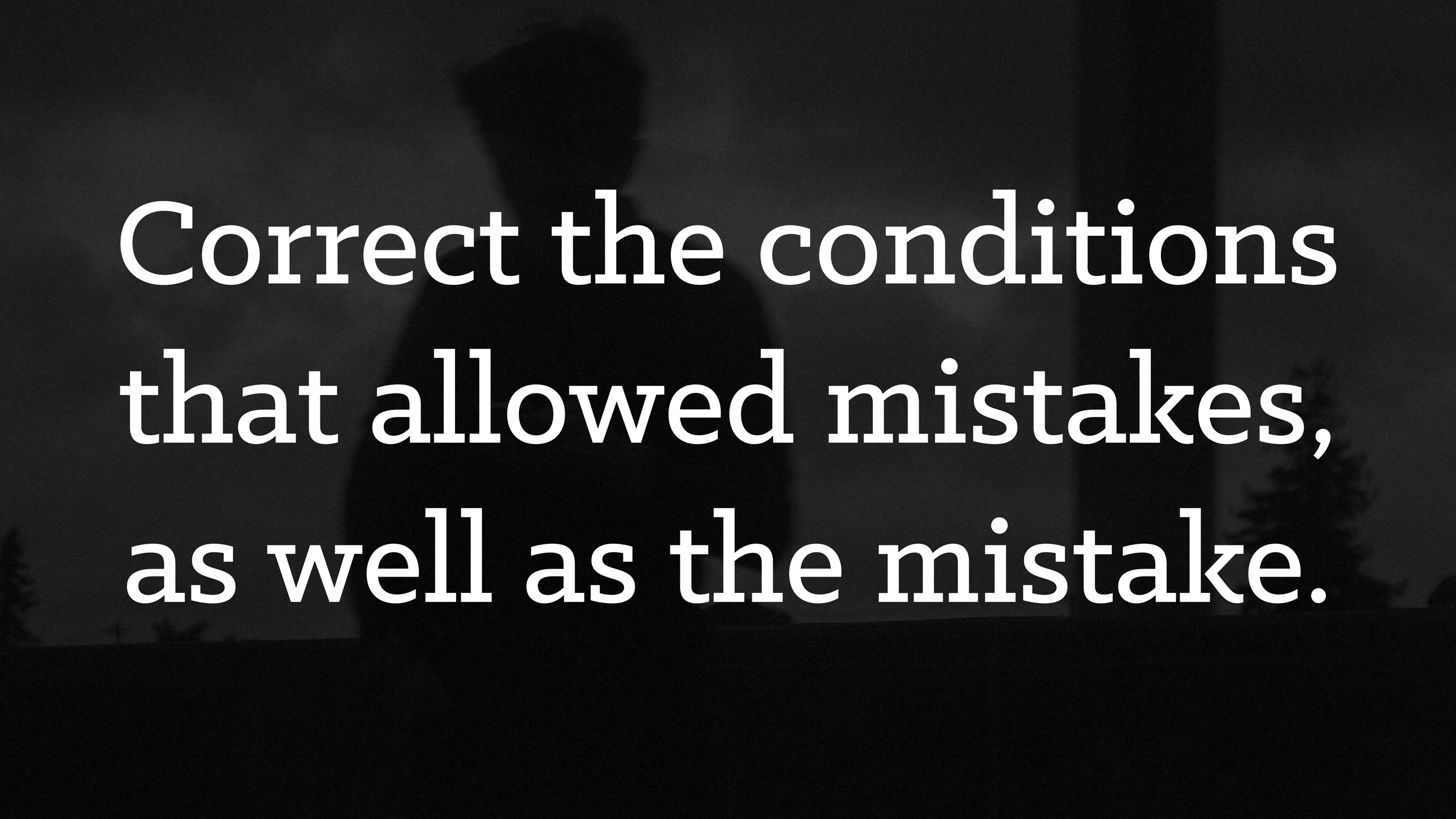
Chevron Refinery

BART ATC

Asiana #214

Therac-25

New Orleans Levee

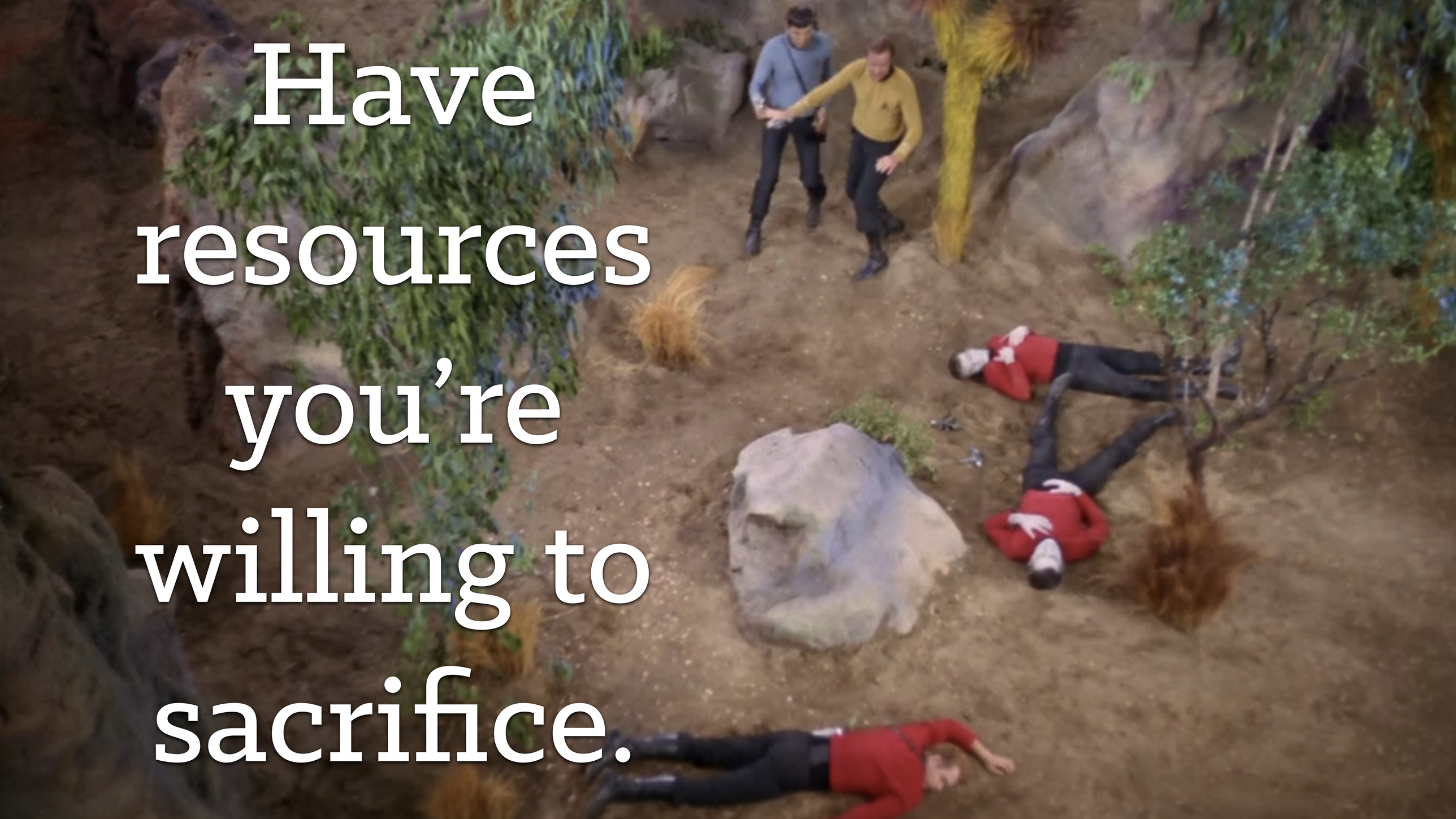

Correct the conditions
that allowed mistakes,
as well as the mistake.


Process is
Priceless


Build flexible
tools for experts.


Separate
Your
Concerns

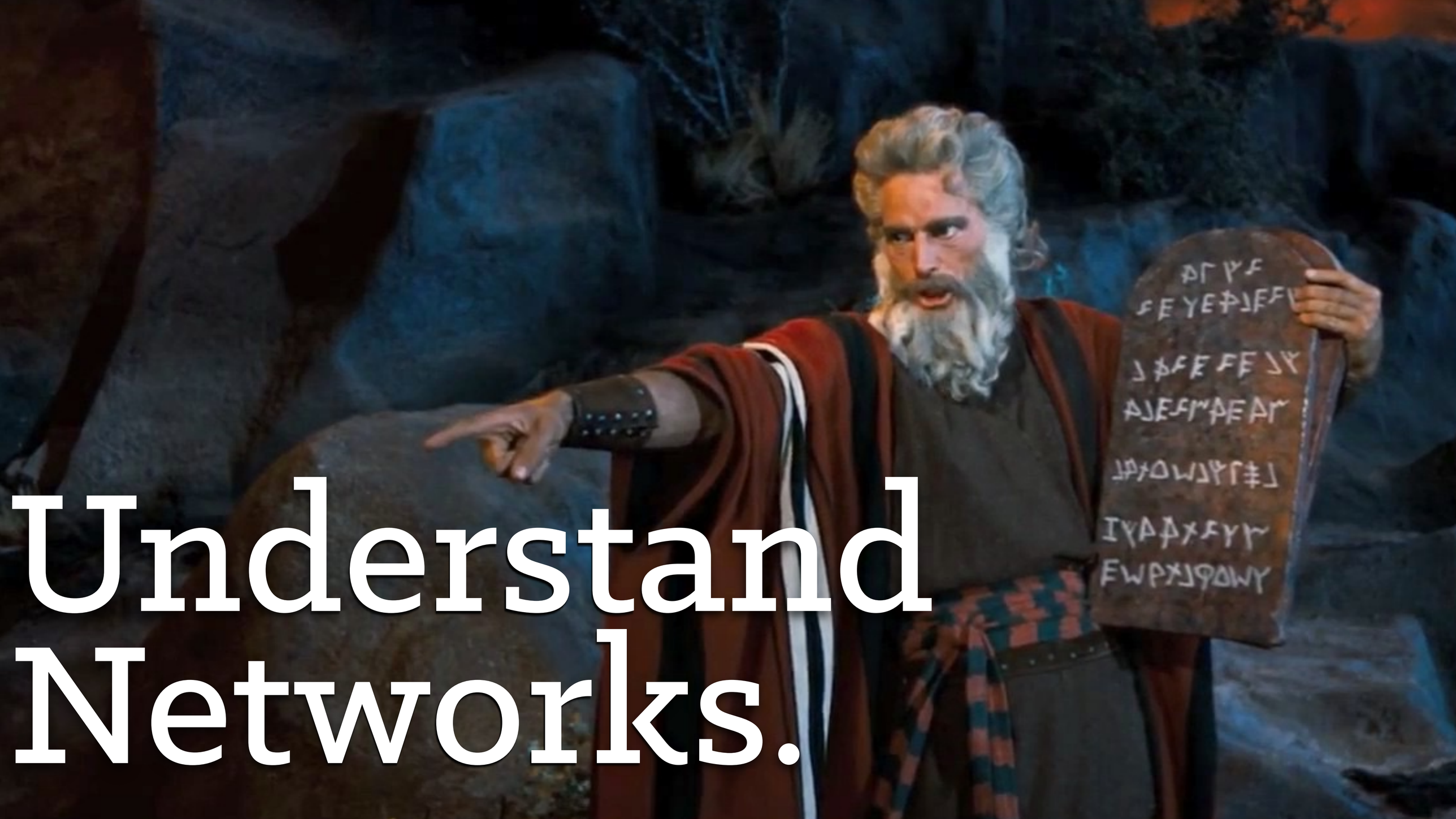

Build with
Failure in mind.


Have
resources
you're
willing to
sacrifice.


Study
accidents.


Every system
carries the
potential for its
own destruction.


Some things aren’t
worth building.



Understand Networks.

0. The network is unreliable.
1. Latency is non-zero.
2. Bandwidth is finite.
3. The network is insecure.
4. Topology changes.
5. There are many administrators.
6. Transport cost is non-zero.
7. The network is heterogenous.


Thanks
so
much!
@bltroutwine

Recommended Reading

“Normal Accidents: Living with High-Risk Technologies”, Charles Perrow

“Digital Apollo: Human and Machine in Spaceflight”, David A. Mindel

“Command and Control: Nuclear Weapons, the Damascus Accident, and the Illusion of Safety”, Eric Schlosser

“Erlang Programming”, Simon Thompson and Francesco Cesarini

“Steeltown, USSR”, Stephen Kotkin

“The Truth About Chernobyl”, Grigorii Medvedev

“Real-Time Systems: Design Principles for Distributed Embedded Applications”, Hermann Kopetz

“The Apollo Guidance Computer: Architecture and Operation”, Frank O’Brien

“Why Do Computers Stop and What Can Be Done About It?”, Jim Gray

“Thirteen: The Apollo Flight That Failed”, Henry S.F. Cooper Jr.