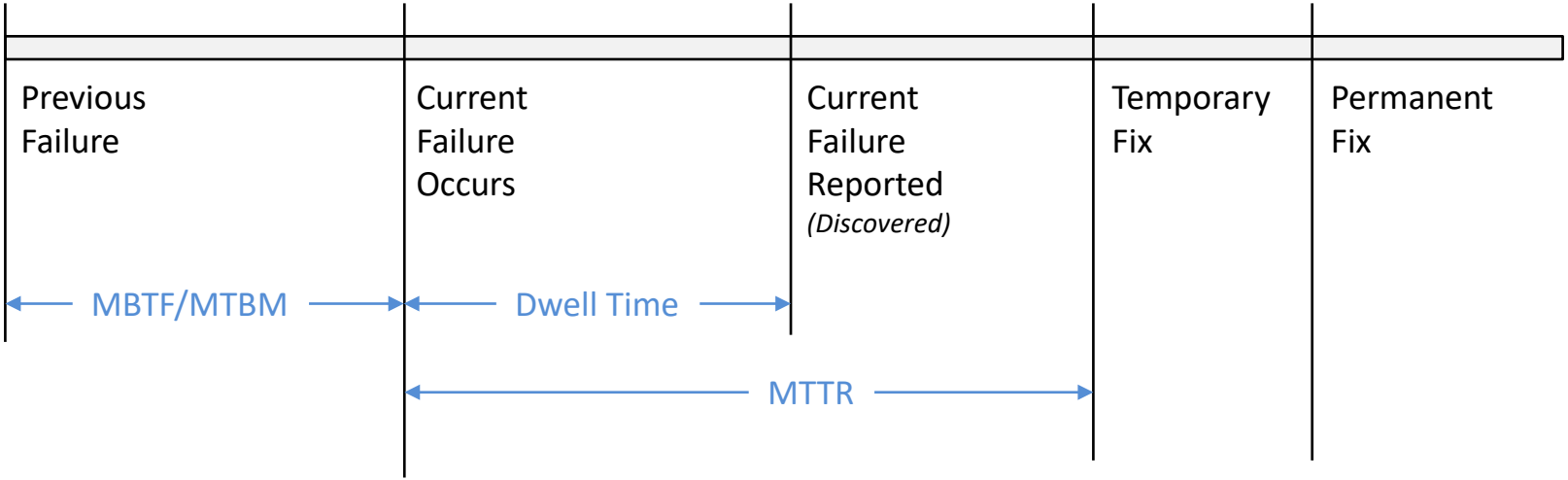
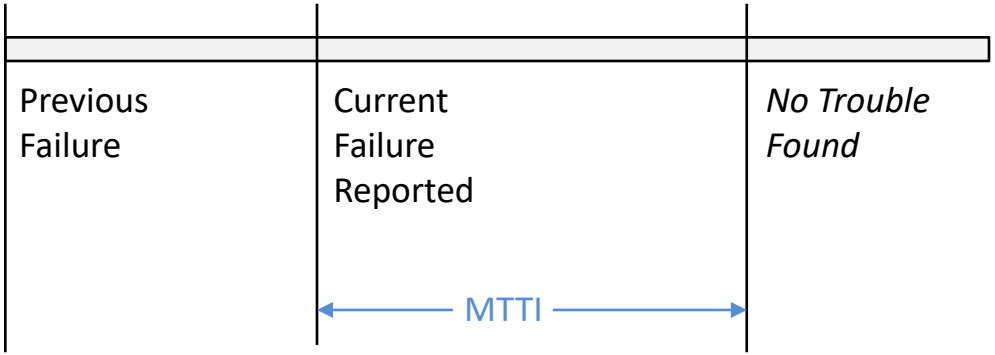
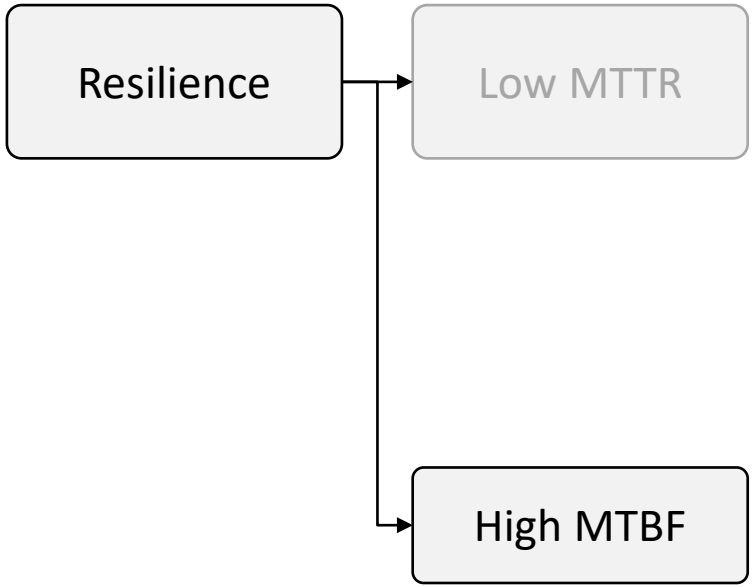
- Foundations
- Understanding the System
  - Components
  - Models
  - System Knowledge
- Process
  - The Half Split Method
- Some Examples

RULE11·TECH

the HEDGE
@rule11.tech

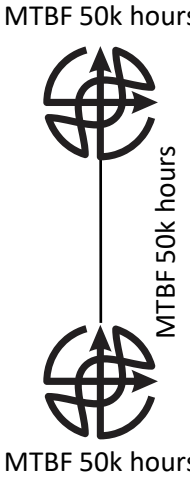

Previous Failure
Current Failure Occurs
Current Failure Reported
(Discovered)
Temporary Fix
Permanent Fix
MBTF/MTBM
Dwell Time
MTTR


Previous
Failure
Current
Failure
Reported
No Trouble
Found
MTTI


Resilience
Low MTTR
High MTBF

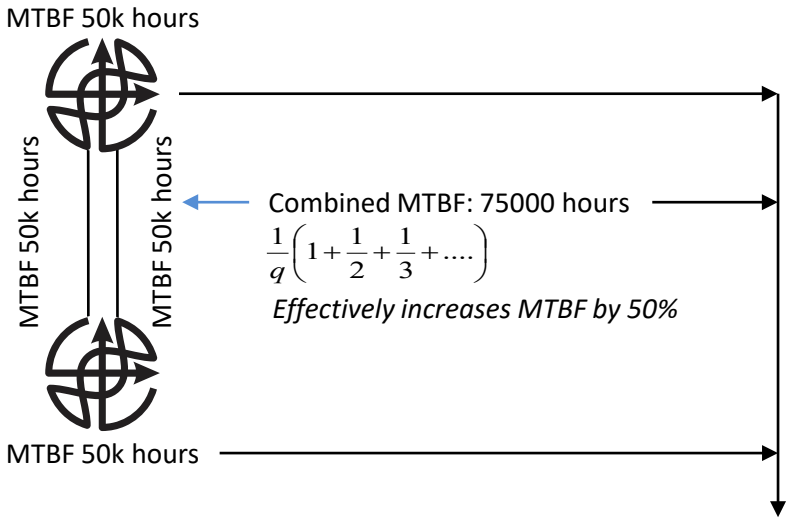
Define “Down”

- A business critical application not operating at all?
- A business critical application operating slowly?
- The vice president cannot get to his email?
- An entire site is off line?
  - How large of a site?
- The president’s child downloaded a virus onto his personal computer?



$$\frac{1}{\frac{1}{MTBF_1}+\frac{1}{MTBF_2}+\frac{1}{MTBF_3}+....}$$

Combined MTBF: 16667 hours



$$\frac{1}{\frac{1}{MTBF_1}+\frac{1}{MTBF_2}+\frac{1}{MTBF_3}+....}$$

Combined MTBF: 18750 hours

2,5000 2ru 64x100g routers
1,024 paths ToR to ToR

$$a_t = \frac{1}{\frac{1}{a_1} + \frac{1}{a_2} + \ldots}$$

2,5000 2ru 64x100g routers
1,024 paths ToR to ToR

2,500 fabric routers
MTBF 219,000 hours (25 years)

2,500 fabric routers x 64 ports each
160,000 optical units
MTBF 876,000 hours (100 years)

80,000 optical fibers
160,000 optical connectors
MTBF 876,000 hours (100 years)

$$a_t = \frac{1}{\frac{1}{a_1} + \frac{1}{a_2} + \ldots}$$

2,5000 2ru 64x100g routers
1,024 paths ToR to ToR

$$a_t = \frac{1}{\frac{1}{a_1} + \frac{1}{a_2} + \ldots}$$

2,500 fabric routers
MTBF 219,000 hours (25 years)

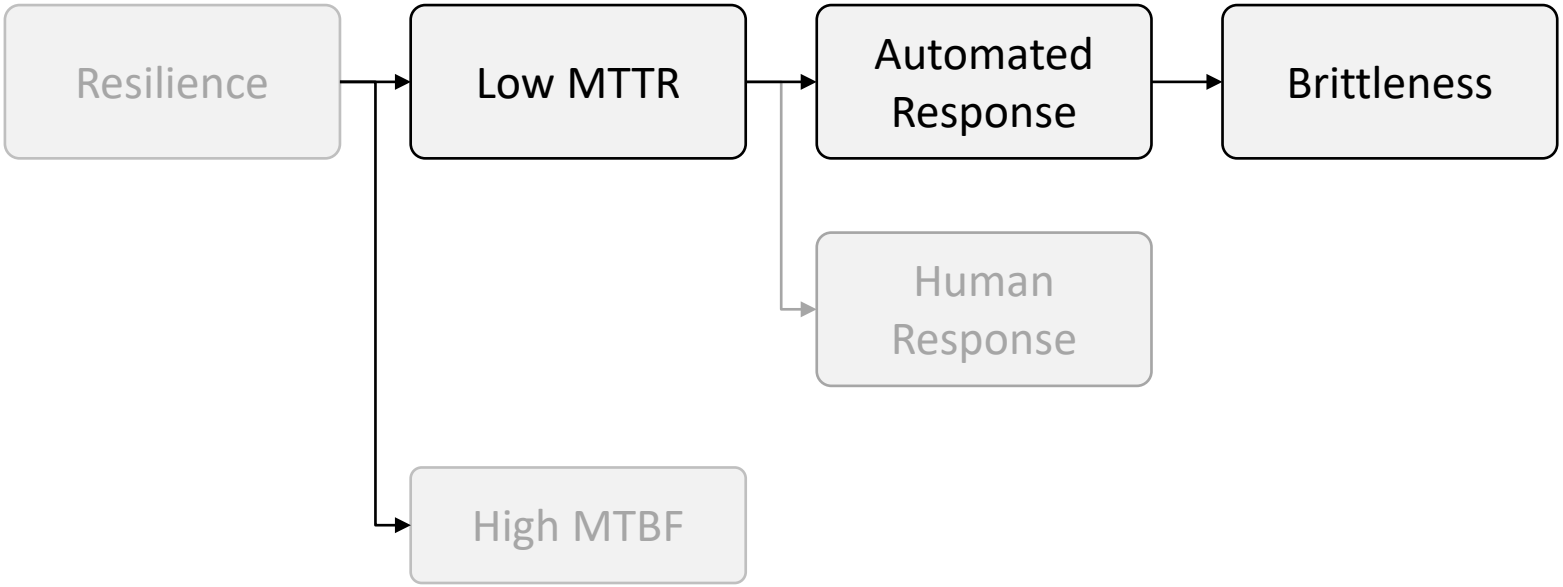
1 router failure every 87.6 hours

2,500 fabric routers x 64 ports each
160,000 optical units
MTBF 876,000 hours (100 years)

1 optical failure every 5.5 hours

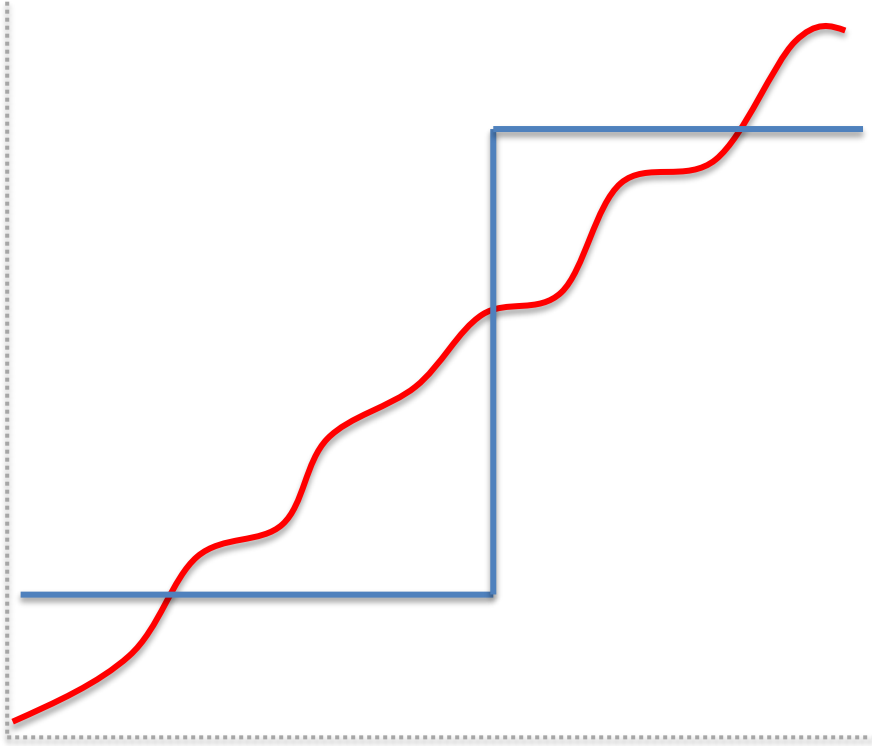
80,000 optical fibers
160,000 optical connectors
MTBF 876,000 hours (100 years)

1 optical failure every 5.5 hours


Resilience
Low MTTR
Automated Response
Brittleness
Human Response
High MTBF

Automated Responses

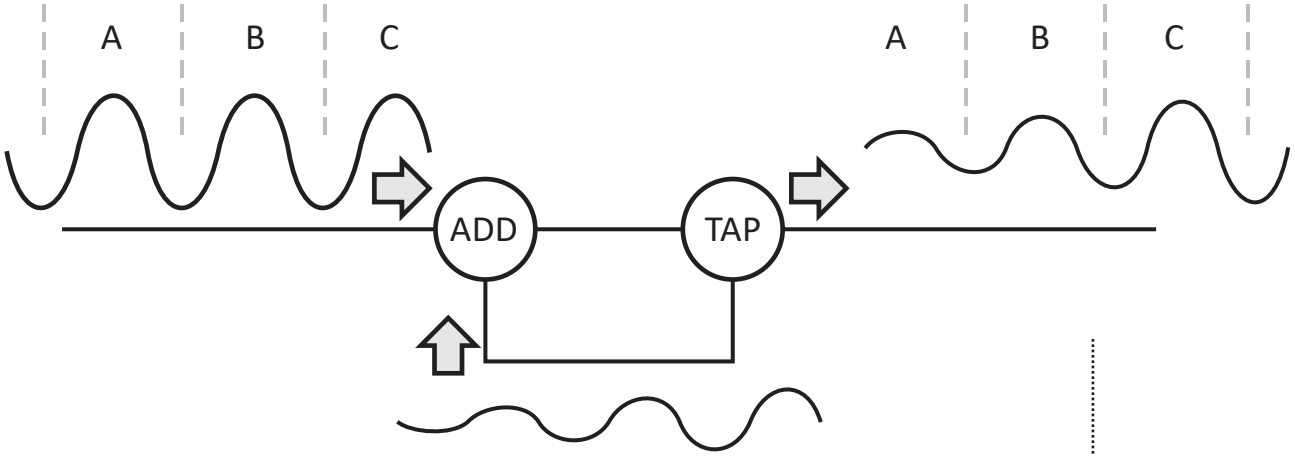
- Automated responses can decrease MTTR
  - When they are well designed
  - When they function as intended
- Automated responses can increase MTTR
  - When they are not well designed
  - When they cause unintended consequences

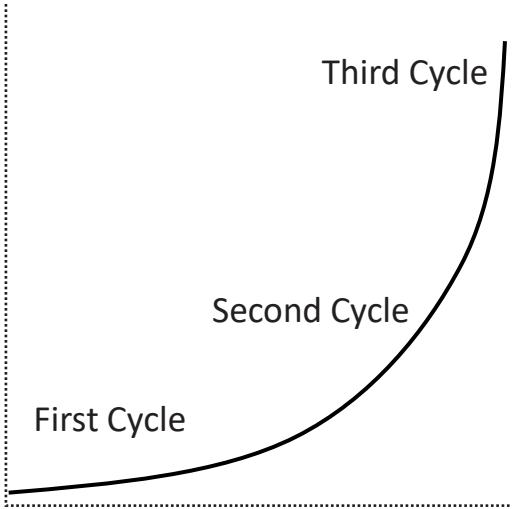


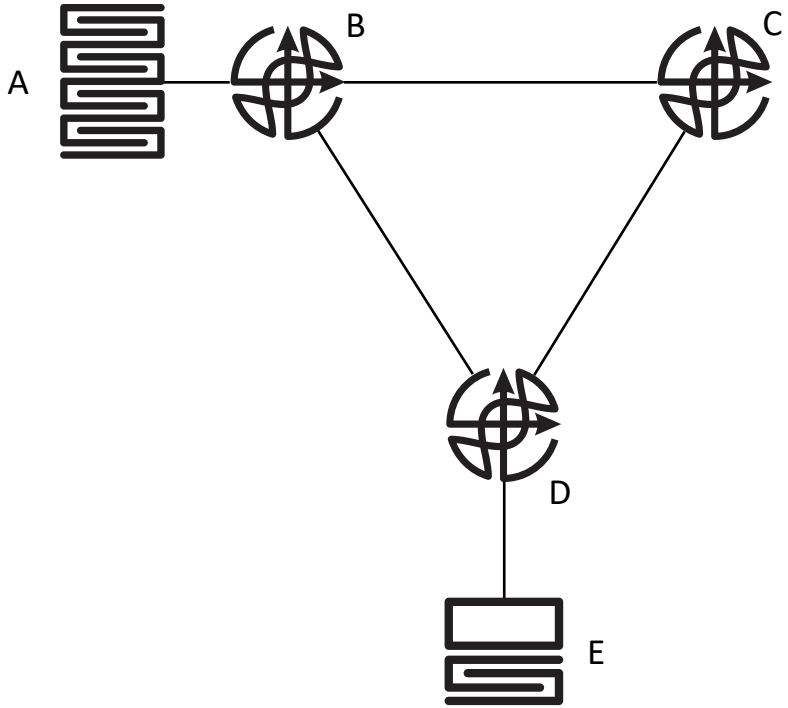
In a complex system, the cumulative effect of a large number of small optimizations is externally indistinguishable from a radical leap.

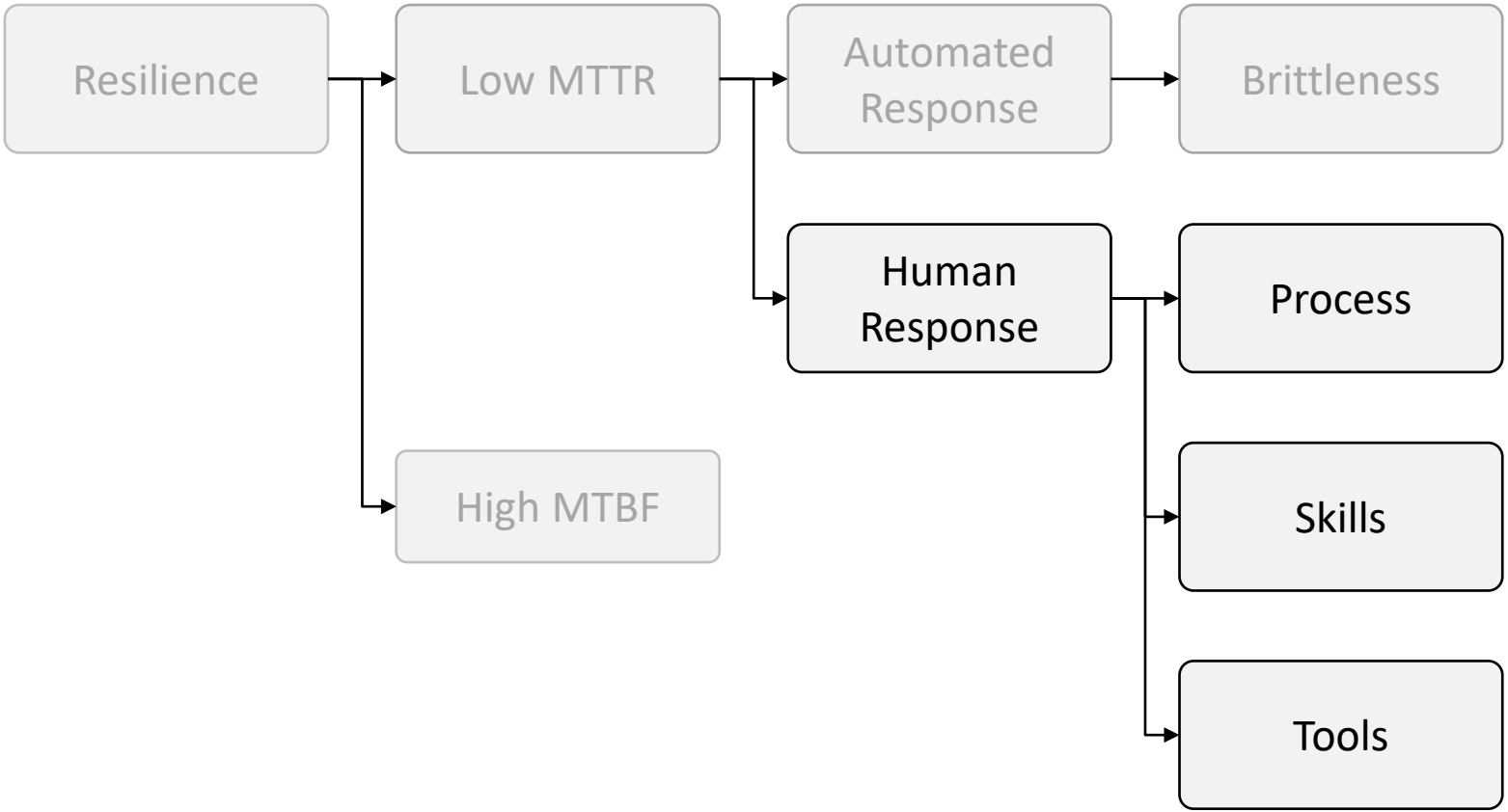
In a complex system, you can know your component in depth and the components around yours at a high level. The rest is rumor and pop psychology

Keith's Law and the first corollary


A
B
C
A
B
C
ADD
TAP


Third Cycle
Second Cycle
First Cycle


A
B
C
D
E


Resilience
Low MTTR
Automated Response
Brittleness
Human Response
Process
High MTBF
Skills
Tools

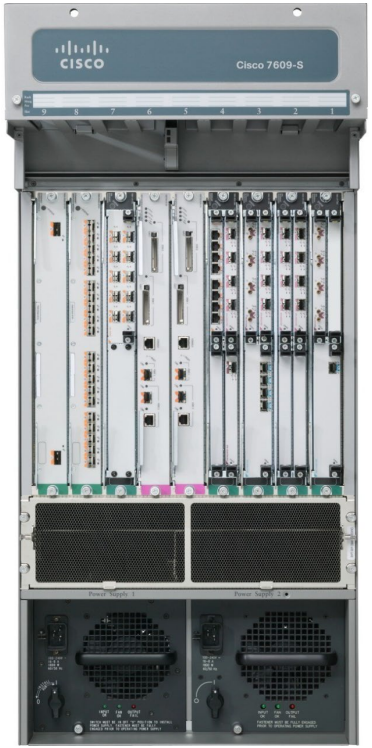
Fixes

- Temp Fix
  - “We know this is going to break again quickly”
  - “We know this increases technical debt”
- Permanent Fix
  - “We know this won’t break anytime soon”
  - “This leaves technical debt where it was before the failure”
  - “This refactors the system and reduces technical debt”
- Technical debt
  - Mismatch between how the system really works and how you think it works

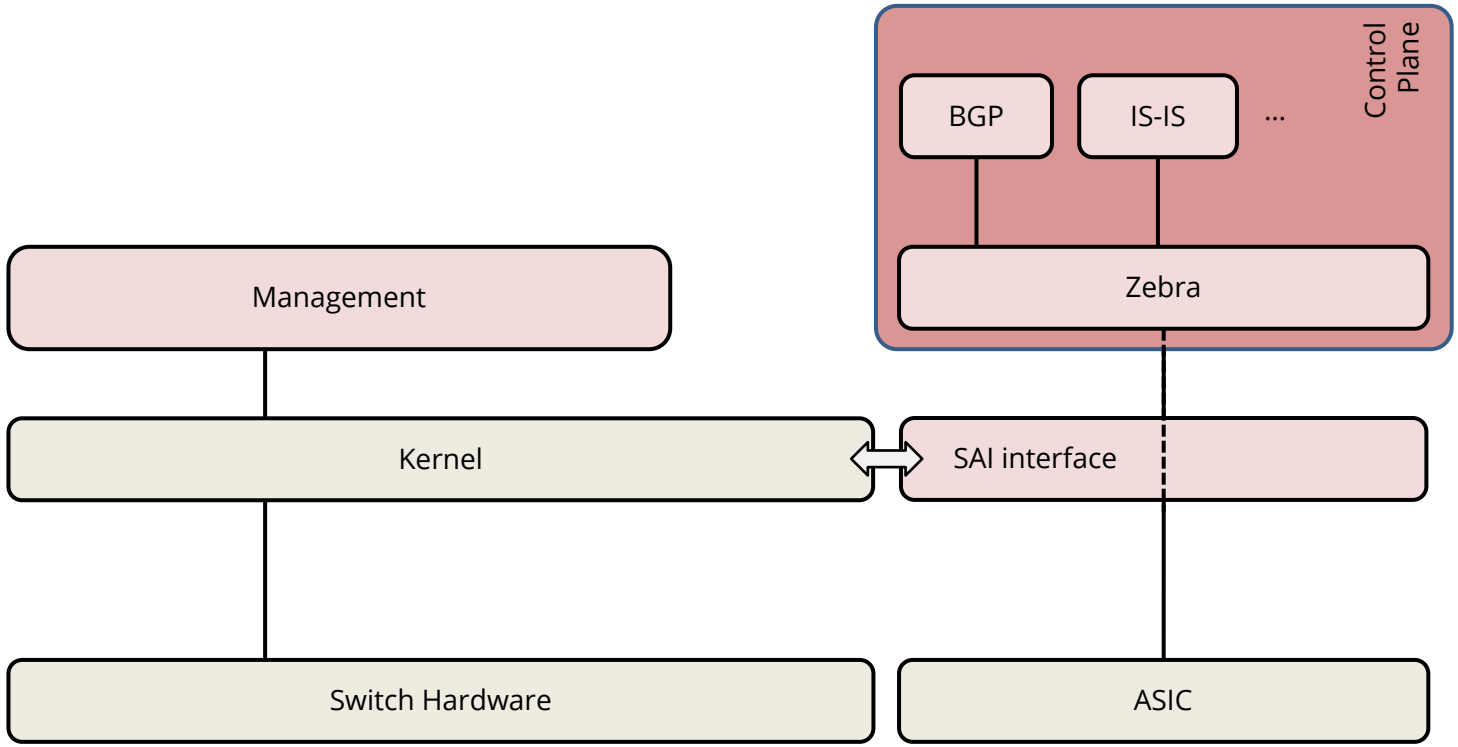
- Components and Abstractions
- Models
- Measuring Things
  - The Heisenberg Problem
  - Manipulability Theory

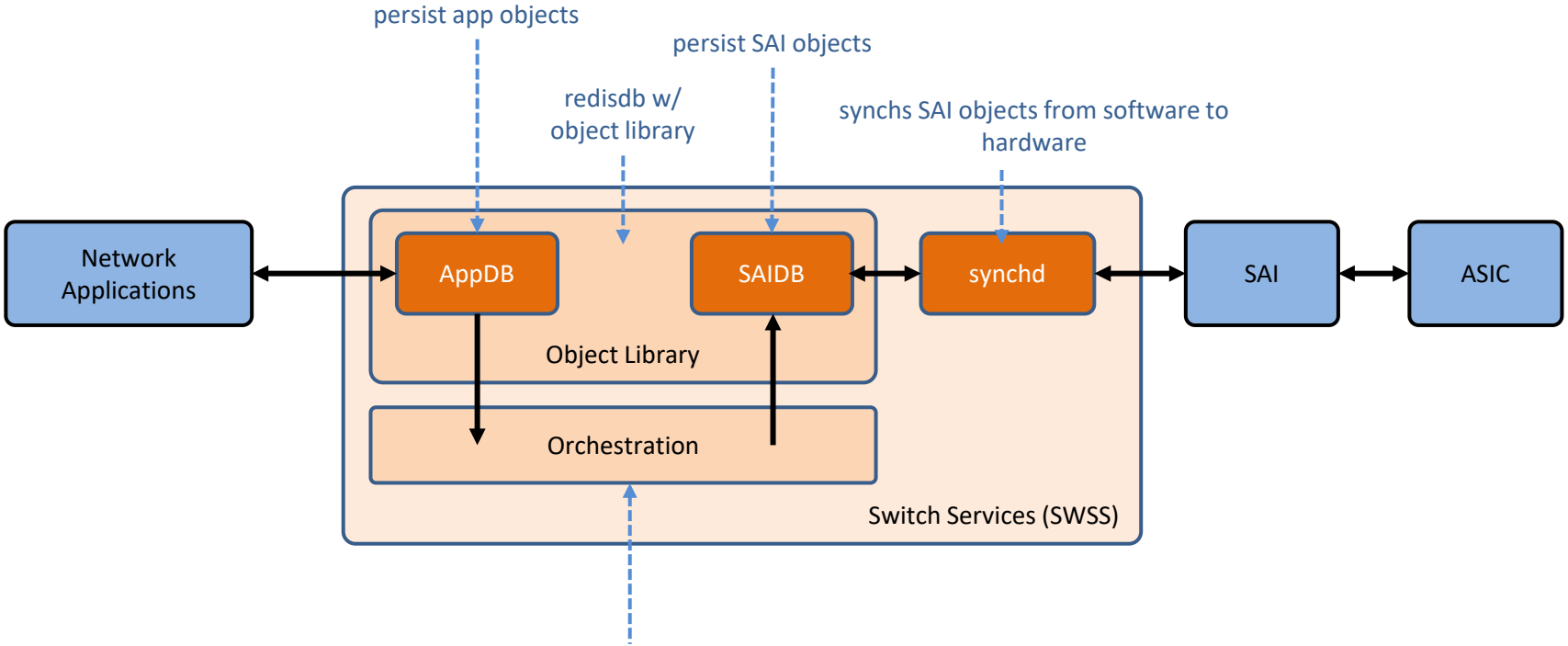
Understand the components

- You see a router, right?
- Is that all you see?



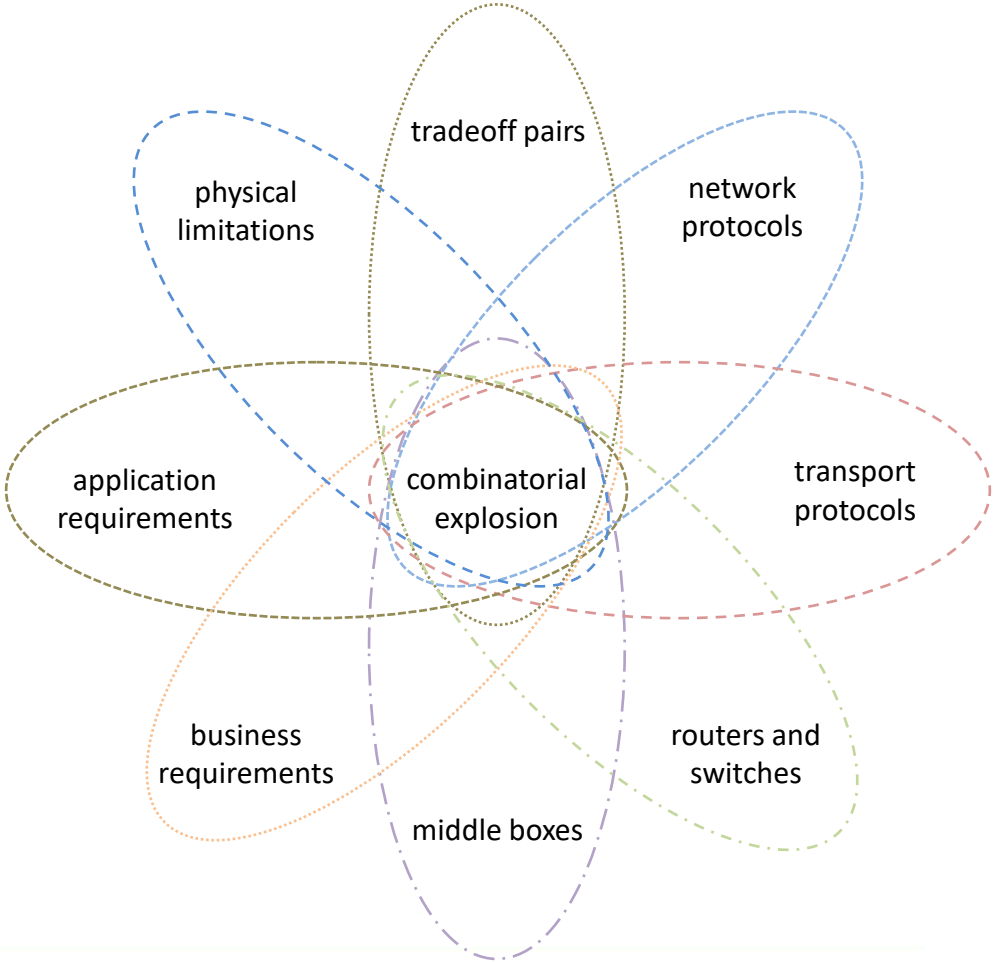
=


Control Plane
BGP
IS-IS
...
Zebra
Management
Kernel
SAI interface
Switch Hardware
ASIC



- translation between apps and SAI objects
- resolution of dependency and conflict


PHY
rx ring
switching asic
switching fabric
output queue
tx ring
PHY


tradeoff pairs
physical limitations
network protocols
application requirements
combinatorial explosion
transport protocols
business requirements
routers and switches
middle boxes

Abstract!

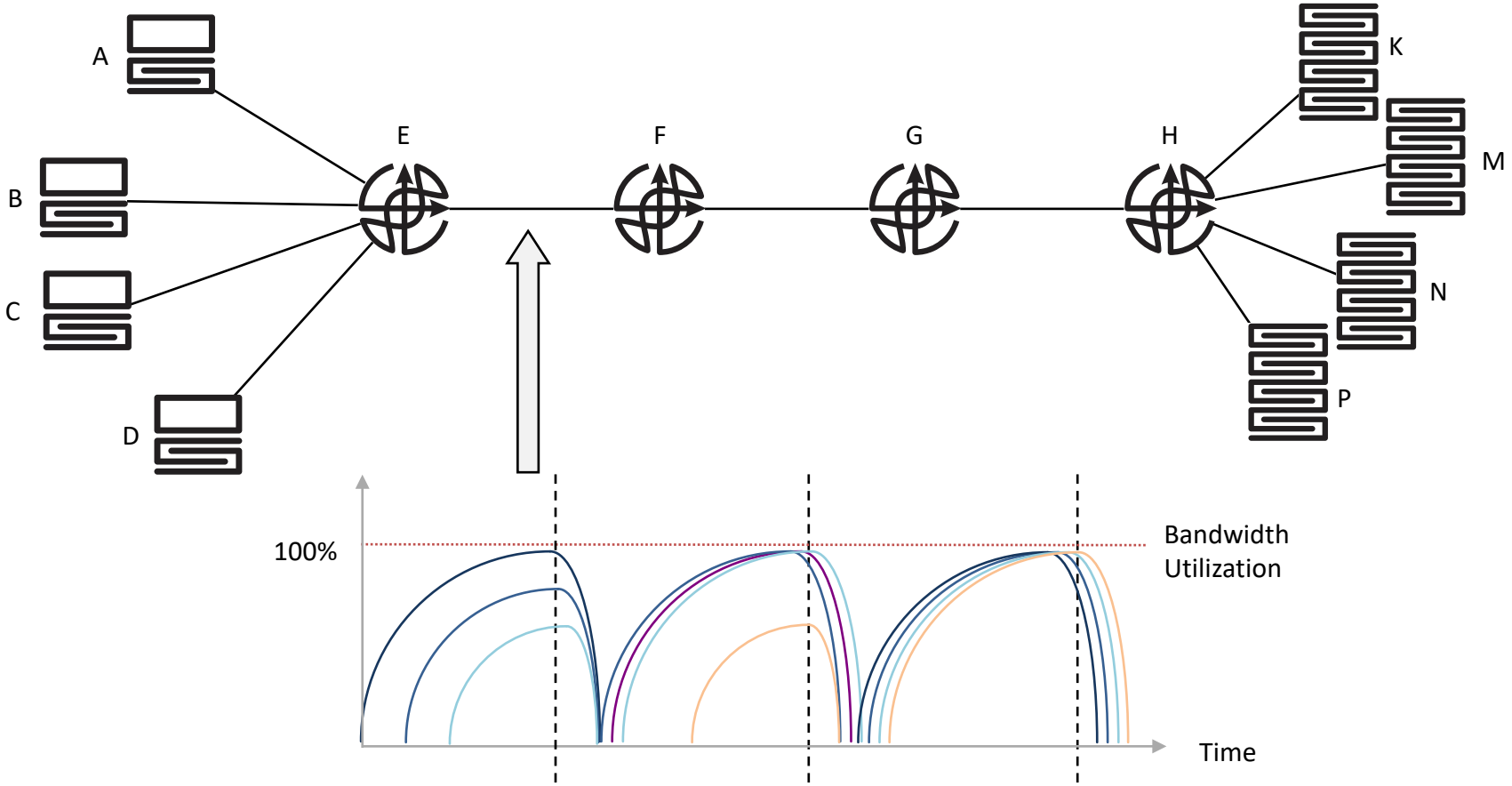
Abstract!

Abstraction hide details

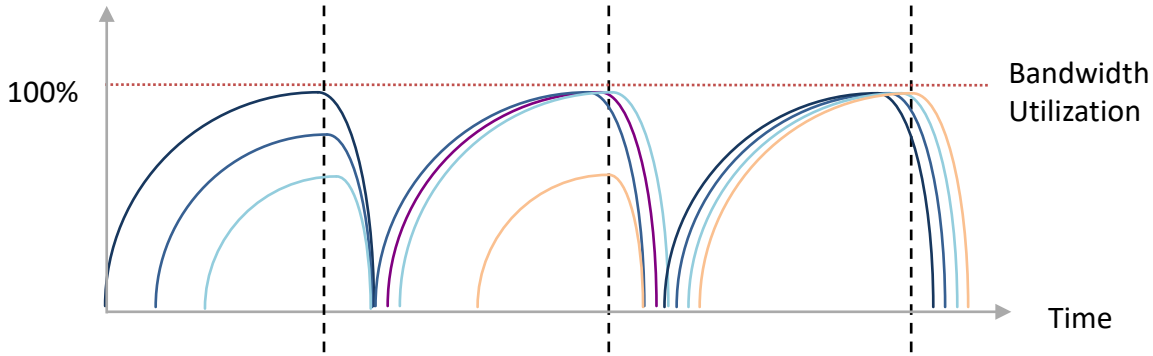
Abstract!

Abstraction hide details

Abstraction leak



each colored line in the graph represents a single flow



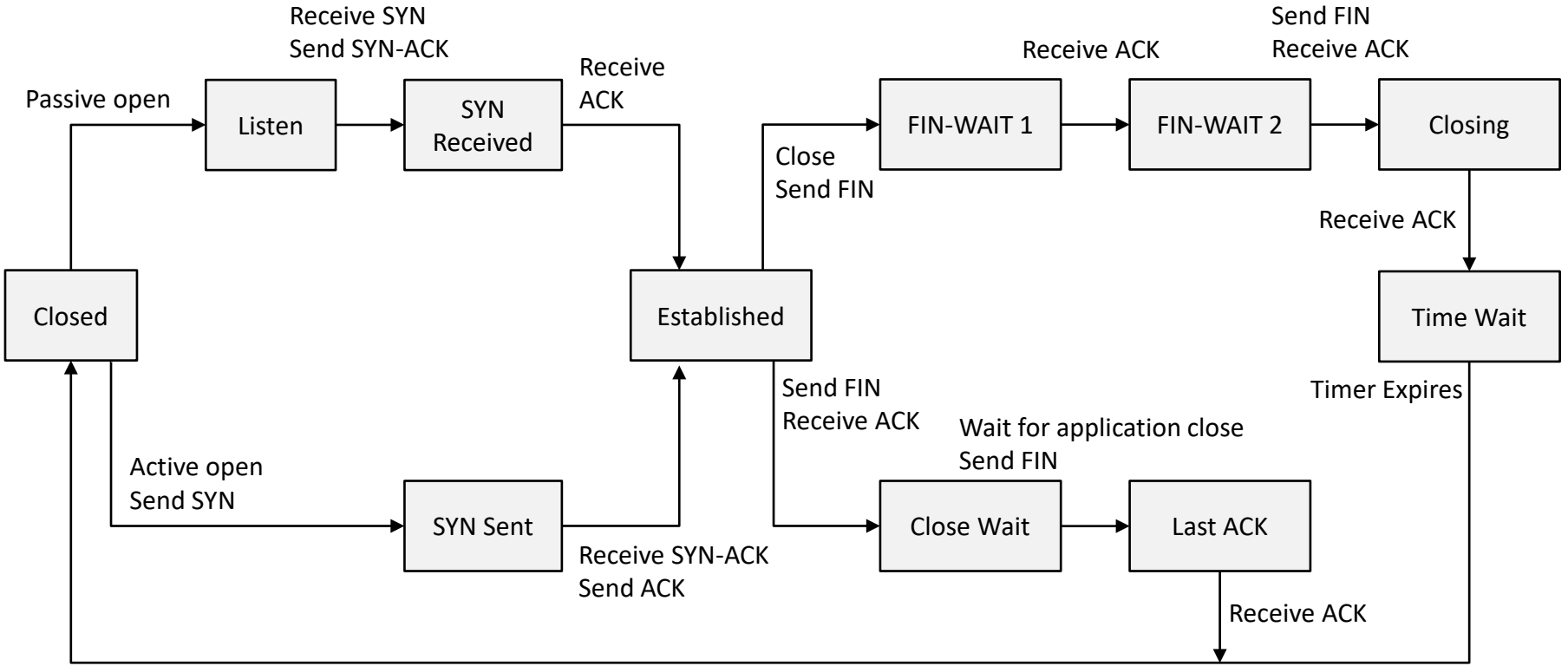
each colored line in the graph represents a single flow

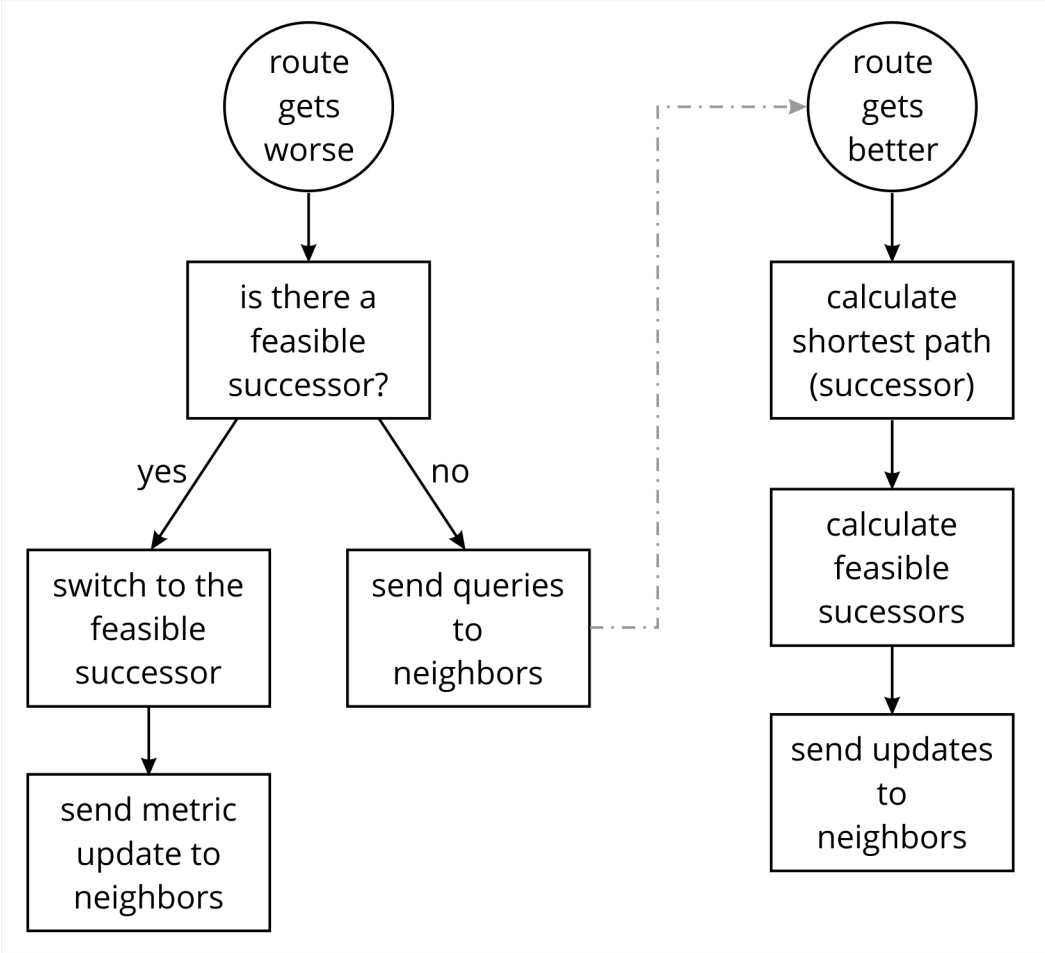
Models

- Models are a form of abstraction
- What models
  - What does this do?
  - What happens when … ?

Models

- Models are a form of abstraction
- What models
  - What does this do?
  - What happens when … ?
- How/why models
  - How does this happen?
  - Why does that happen this way?
- Some examples will help make sense of the differences


Receive SYN
Send SYN-ACK
Receive ACK
Send FIN
Receive ACK
Passive open
Listen
SYN Received
Receive ACK
FIN-WAIT 1
FIN-WAIT 2
Closing
Close
Send FIN
Receive ACK
Closed
Established
Time Wait
Send FIN
Receive ACK
Timer Expires
Wait for application close
Send FIN
Active open
Send SYN
SYN Sent
Close Wait
Last ACK
Receive SYN-ACK
Send ACK
Receive ACK


route gets worse
is there a feasible successor?
yes
no
switch to the feasible successor
send metric update to neighbors
send queries to neighbors
route gets better
calculate shortest path (successor)
calculate feasible sucessors
send updates to neighbors

B.G.P

Boundary Gateway Protocol

block marker — 2 bytes
block length — 1 byte
version number — 1 byte
block type — 1 byte (~~keepalive msg~~)
holddown timer — 2 bytes (minutes)

types: open - 1
update - 2
notification - ~~3~~ 4
keepalive - 8

version is currently 1

open: my AS # — 2 byte
link type — 1 byte
- up - 1
- down - 2
- internal - 4
- H-link - 8

(not used in update direction field)

auth type code — 1 byte
- 0 - none

authentication — variable

update: network # — 4 bytes
first hop gateway — 4 bytes
metric — 2 bytes
count of AS — 1 byte
{ direction — 1 byte
{ AS # — 2 byte
} repeat "count" times

repeat structure according to block length

notification: ~~opcode~~ — 2 bytes
data — variable

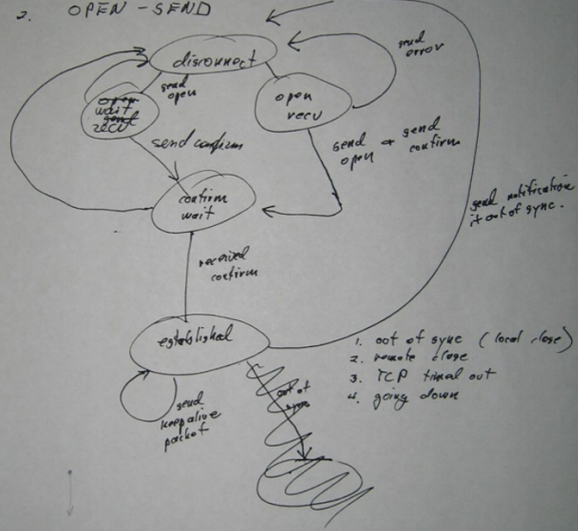
opcodes

1. link type error in open
   - my view of correct link type (1 byte)
2. unknown auth type code
   - no data
3. authentication failure (no data)
4. update error - ~~data is block in error~~
   ~~routing loop in update~~
   ~~two phase error in update~~
   data is subcode (1byte) followed by update block in question (1 network only)

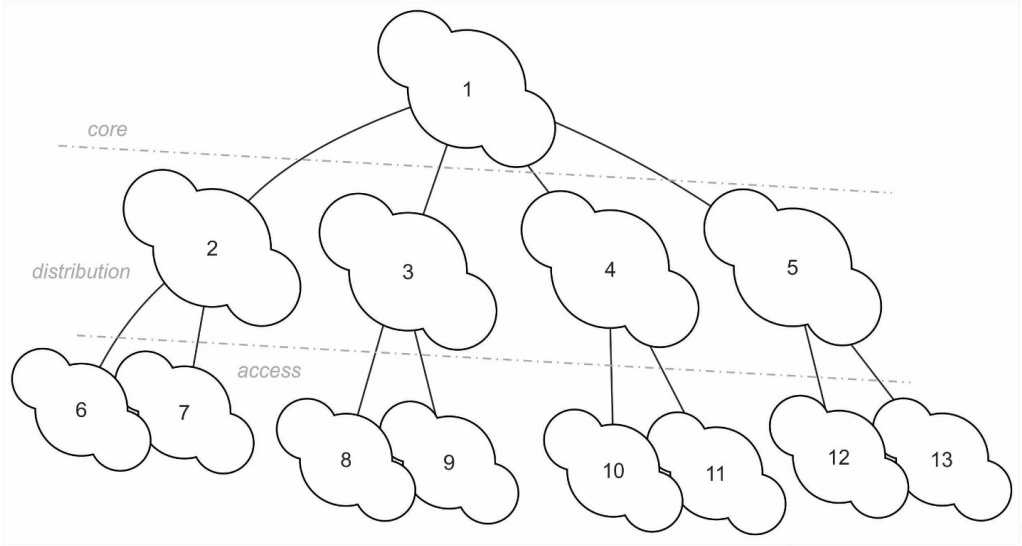
   subcodes - 1 invalid network field
   2. invalid first hop gw
   3. invalid direction code
   4. invalid AS
   5. routing loop
   6. two-phase error
5. connection out of sync - data is last block received (TCP close after packet sent)
6. open confirmed
7. invalid block ~~type~~ type (data is 1 byte block type)
8. invalid version number (data is 1 byte version)

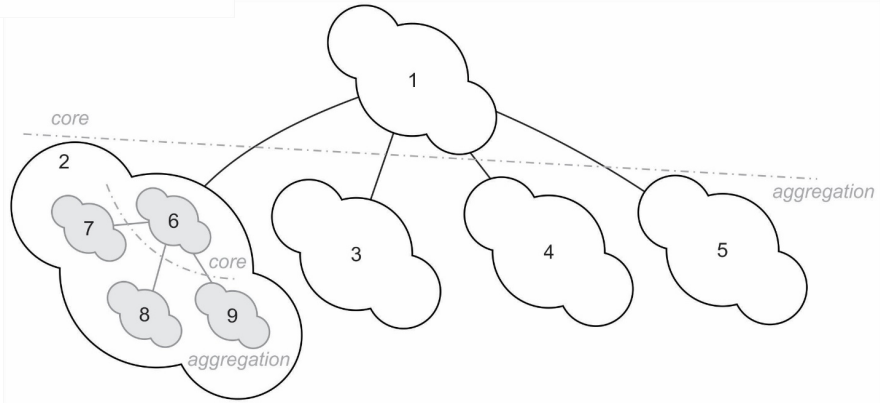
State Diagram

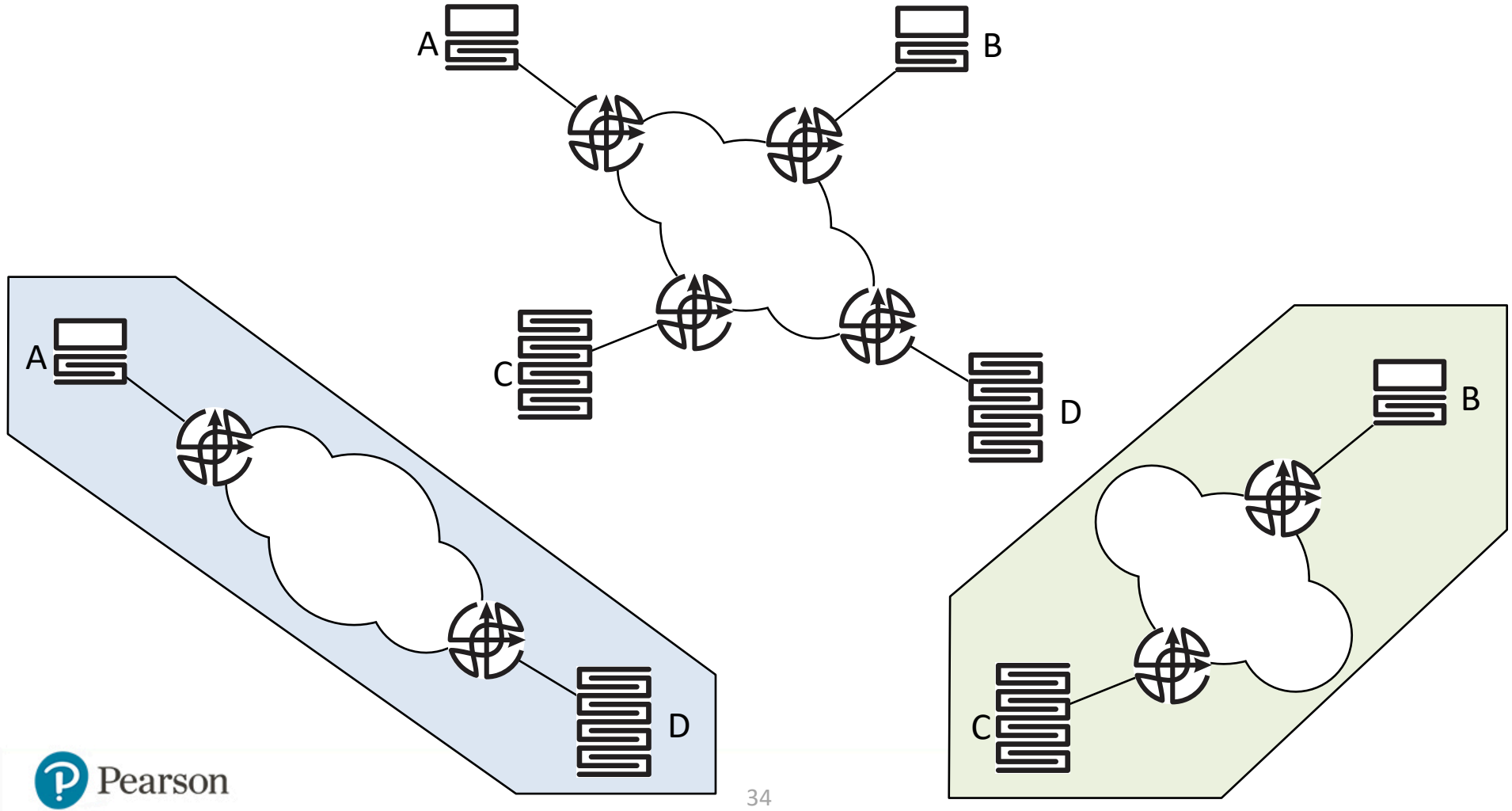
1. initial state is DISCONNECT
2. OPEN - SEND

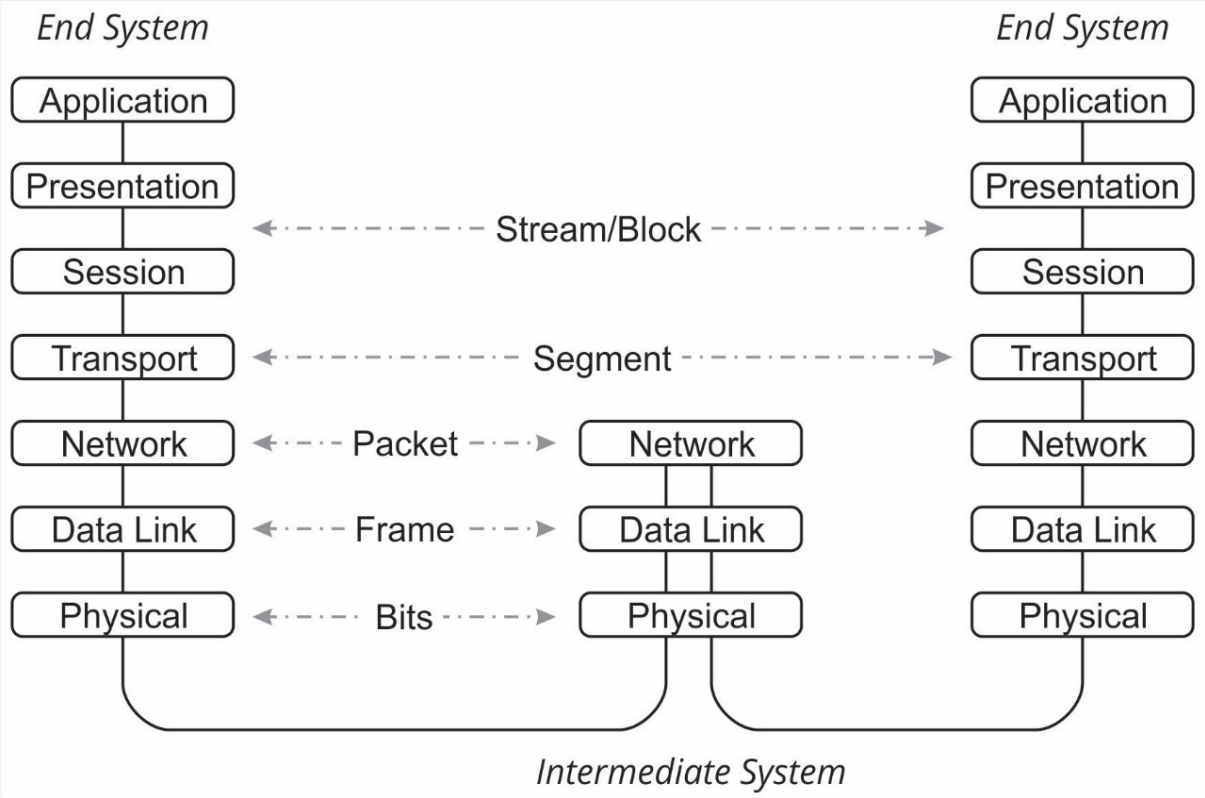


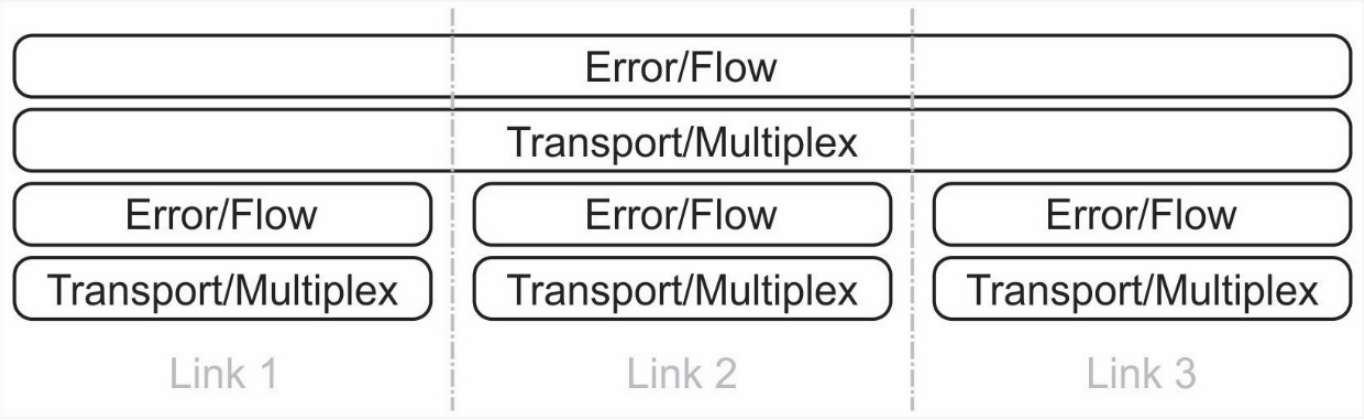
loughead@cisco.com 415-326-1941 (11-7) PST
YAKOV@IBM.COM (914) 945-3896 (8-5) EST


core
distribution
access
1
2
3
4
5
6
7
8
9
10
11
12
13


core
aggregation
core
aggregation
1
2
3
4
5
6
7
8
9

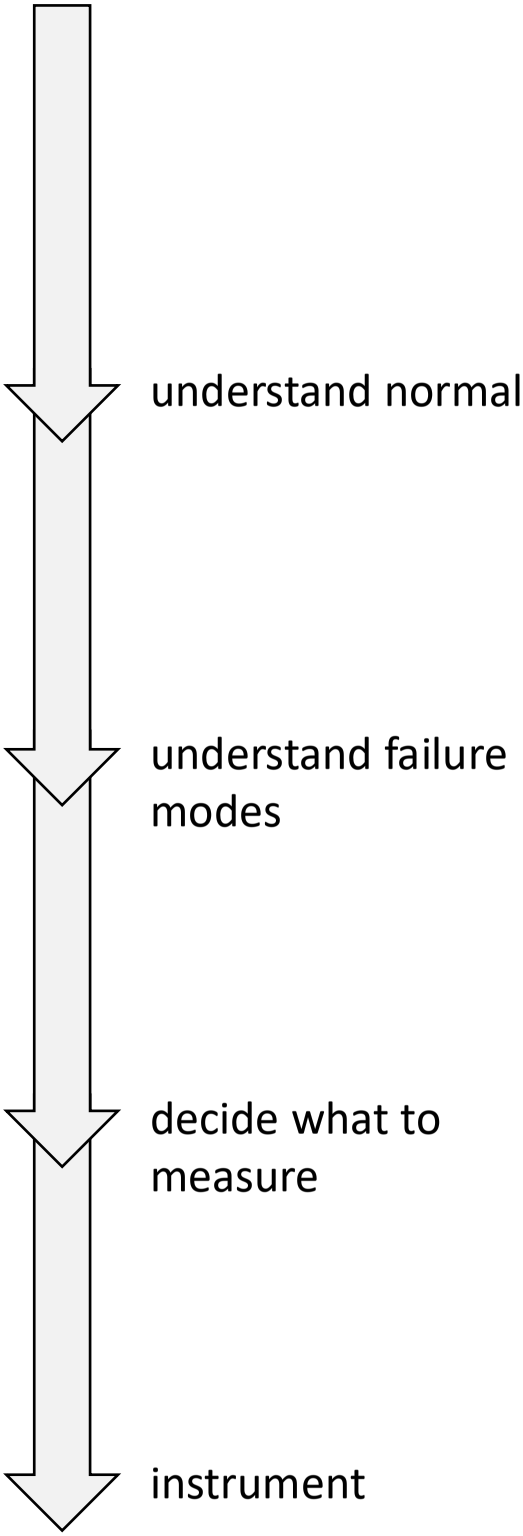

A
B
C
D
A
D
B
C

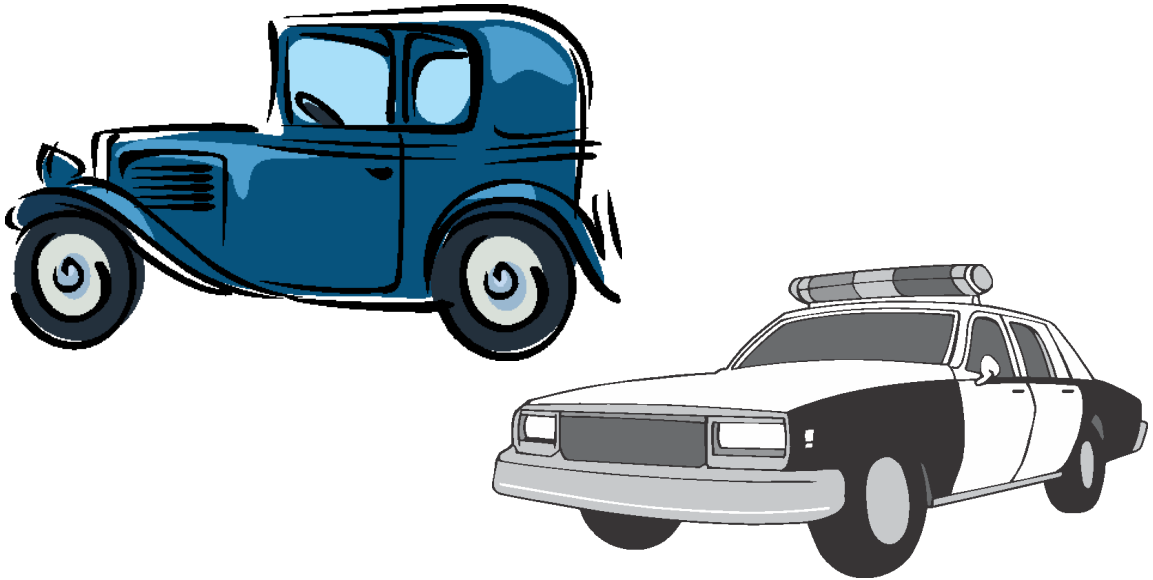

End System
End System
Application
Presentation
Session
Transport
Network
Data Link
Physical
Stream/Block
Segment
Packet
Frame
Bits
Network
Data Link
Physical
Application
Presentation
Session
Transport
Network
Data Link
Physical
Intermediate System


Error/Flow
Transport/Multiplex
Error/Flow
Error/Flow
Error/Flow
Transport/Multiplex
Transport/Multiplex
Transport/Multiplex
Link 1
Link 2
Link 3

Measurement Principles

- Measure anything helpful
  - In determining there is a problem
  - In determining where to start troubleshooting
- Do not expect ongoing measurement to provide all the information you need to troubleshoot a problem


understand normal
understand failure modes
decide what to measure
instrument

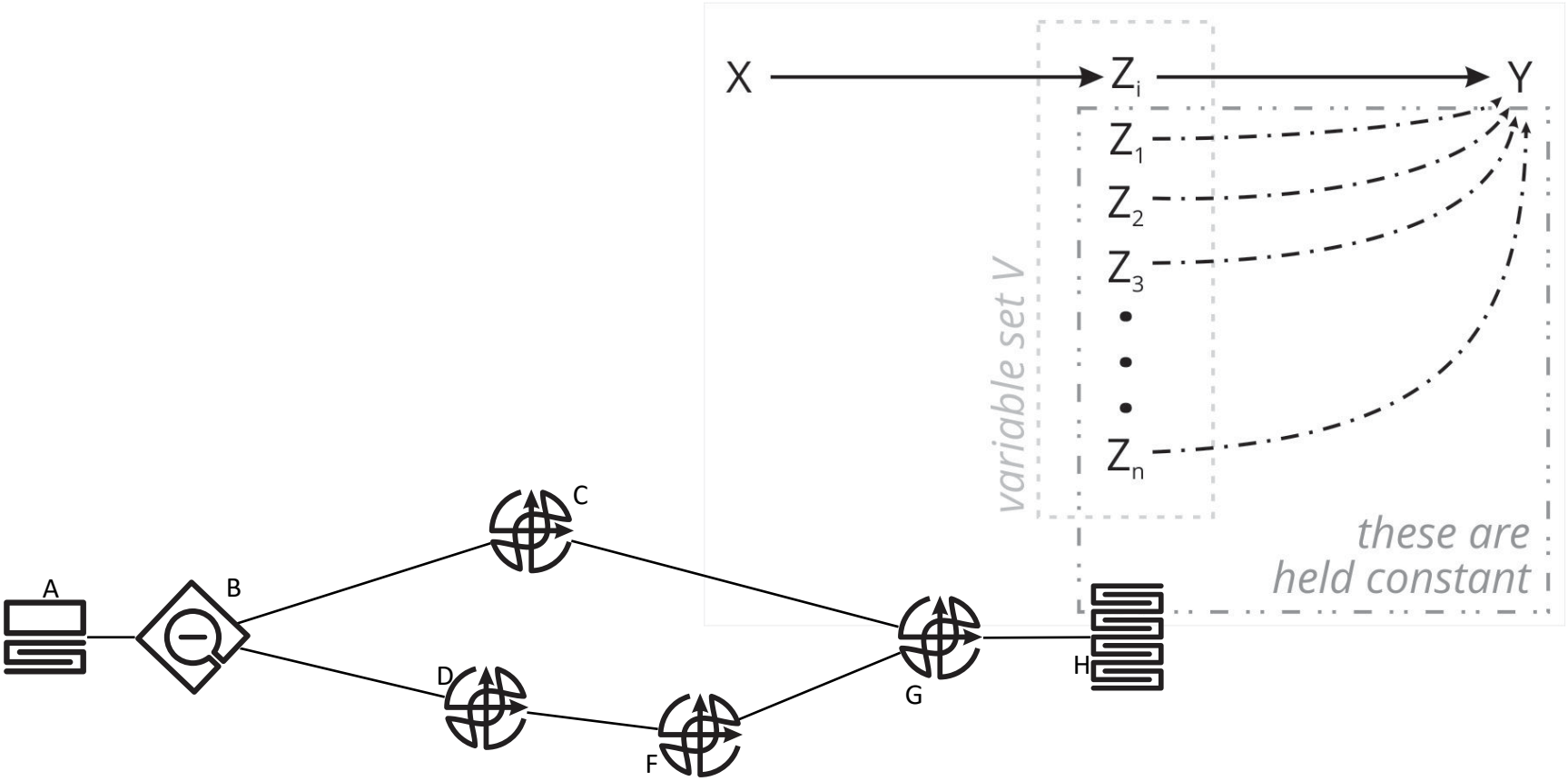


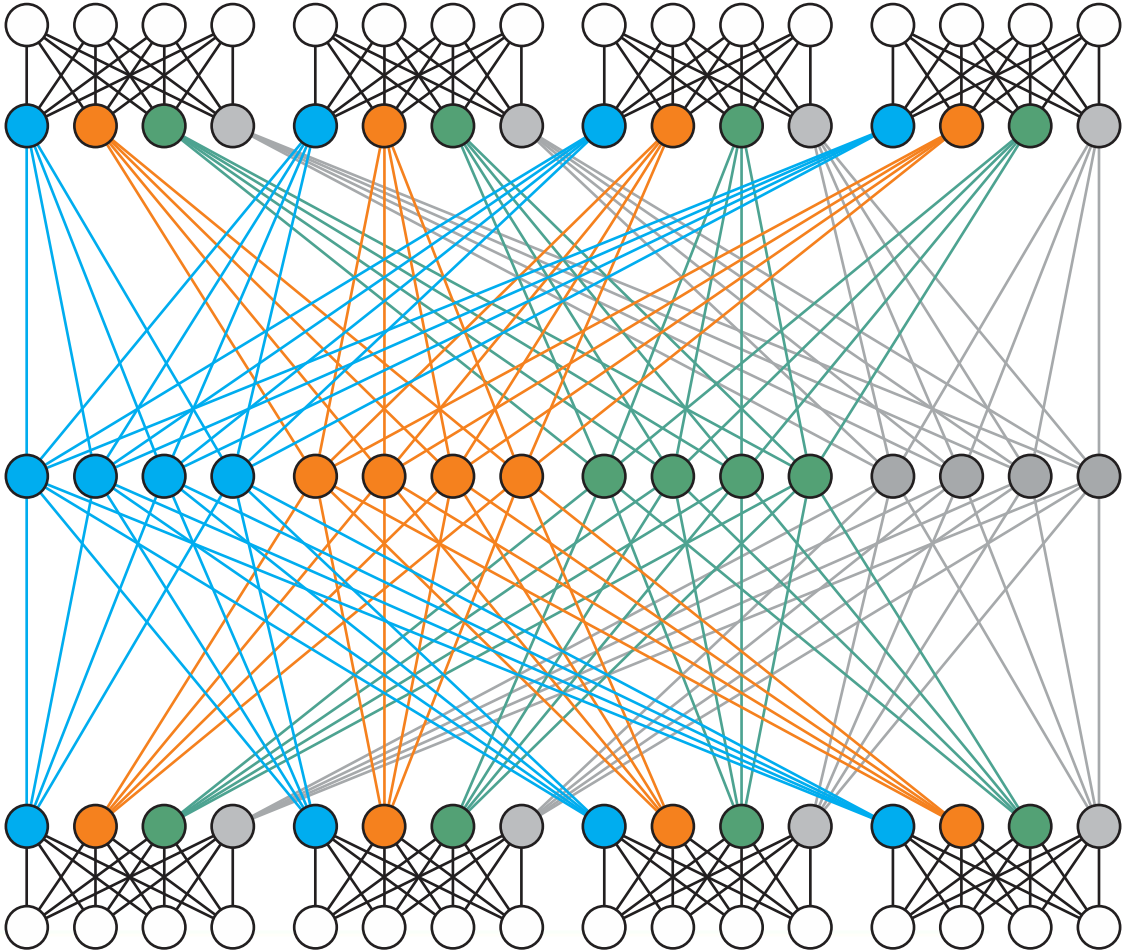
Problems with Observation

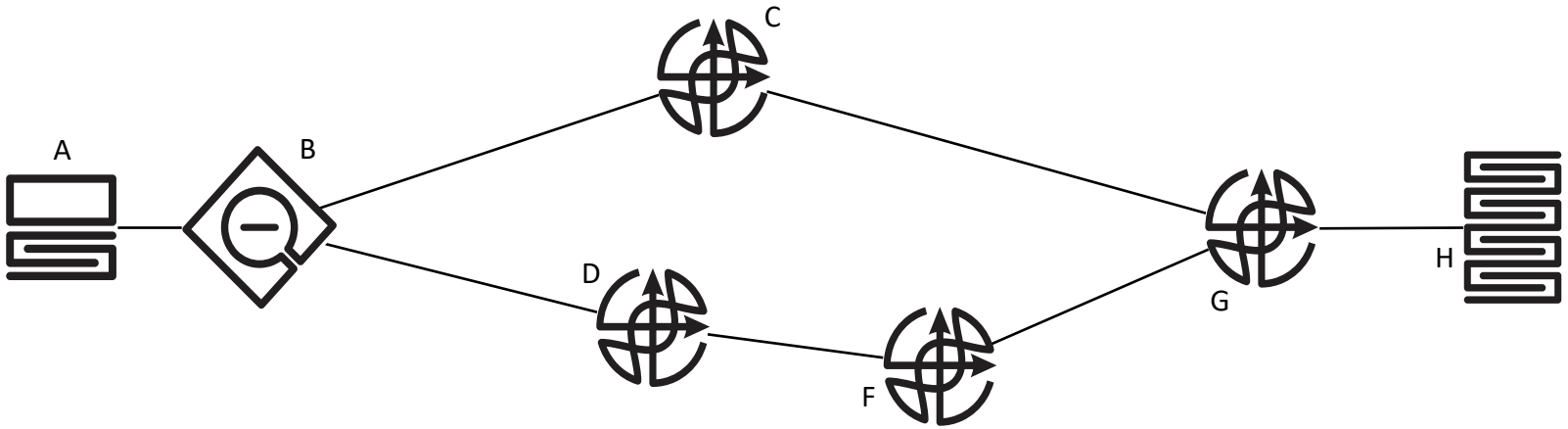
- All these things change the observation
  - Removing links
  - Pinning traffic to a path
  - Showing what is in the queue
  - Just about anything else
- You often cannot observe without changing
- Make certain you take this into account while troubleshooting

Manipulability Theory

- Prove the problem is at “here” before moving deeper into “here” as a system
- Manipulability theory will help
  - A form of causality theory
  - Closely related to the so-called “scientific method”
  - *Making Things Happen* is a good explanation of this theory, for the more philosophically inclined


X
Z_i
Y
Z_1
Z_2
Z_3
Z_n
variable set V
these are
held constant
A
B
C
D
F
G
H





Normal Troubleshooting

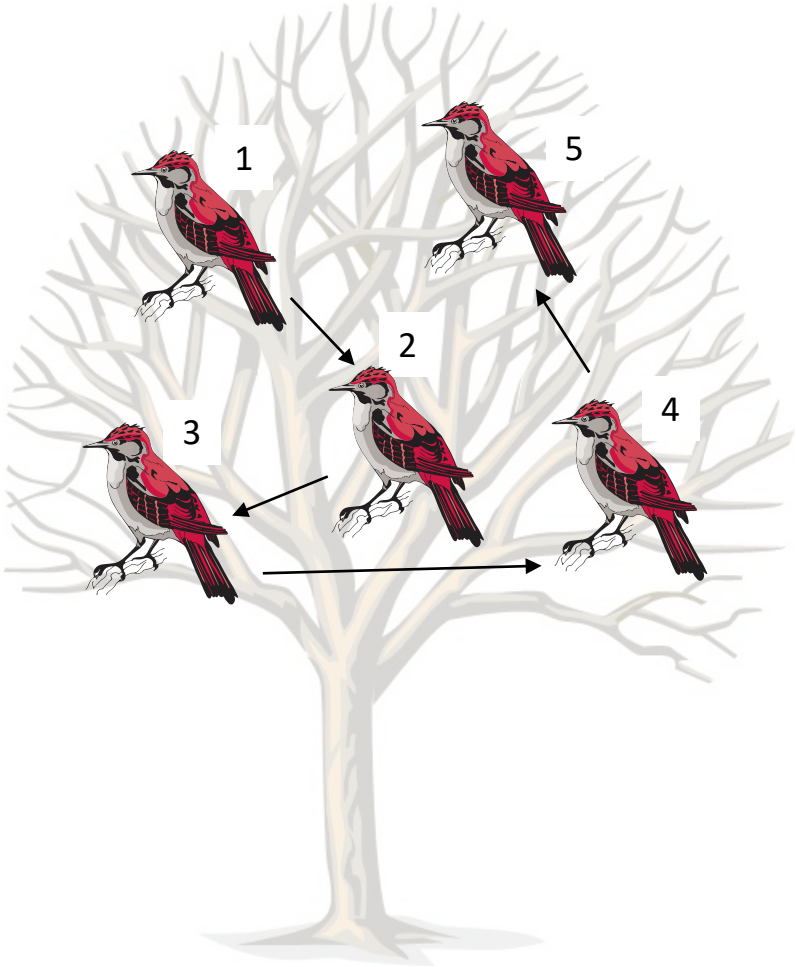
Half Split Method

Some General Notes on Troubleshooting

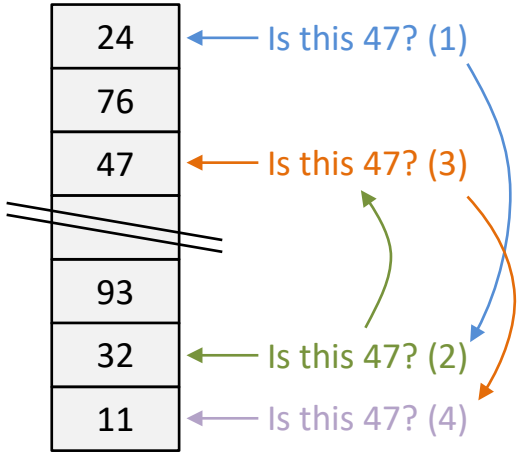
Three diagnostic heuristics were identified as being in use:

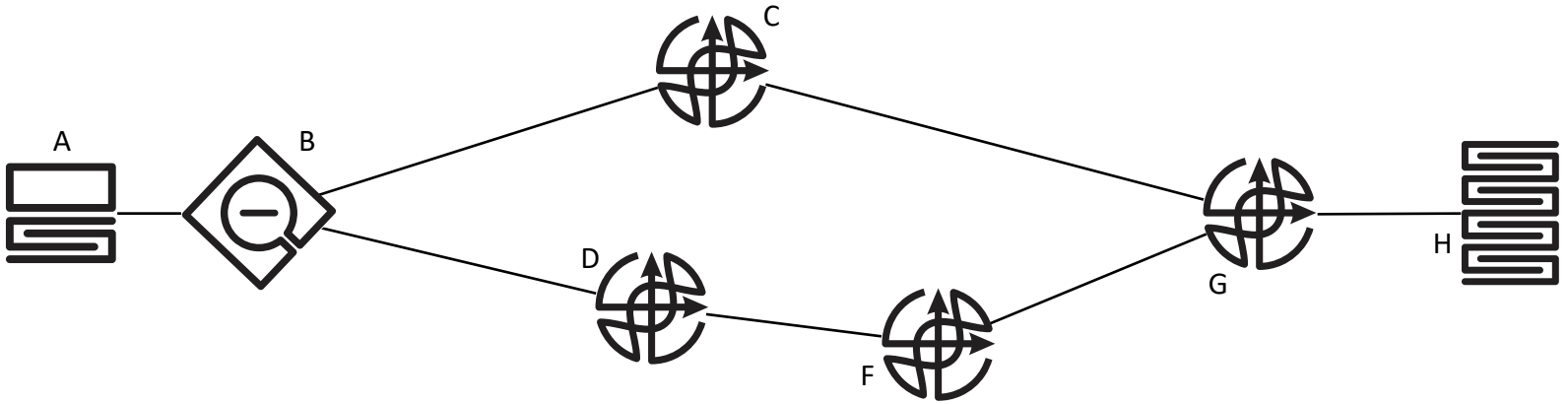
a) initially look for correlation between the behaviour and any recent changes made in the software,
b) upon finding no correlation with a software change, widen the search to any potential contributors imagined, and
c) when choosing a diagnostic direction, reduce it by focusing on the one that most easily comes to mind, either because symptoms match those of a difficult to diagnose event in the past, or those of any recent symptoms match those of a difficult-to-diagnose event in the past, or those of any recent
d) A fourth heuristic is coordinative in nature: when making changes to software in an effort to mitigate the untoward effects or to resolve the issue completely, rely on peer review of the changes more than automated testing (if at all.)

Allspaw, John. "Trade-Offs Under Pressure: Heuristics and Observations of Teams Resolving Internet Service Outages." Lund University, 2015.


1
2
3
4
5

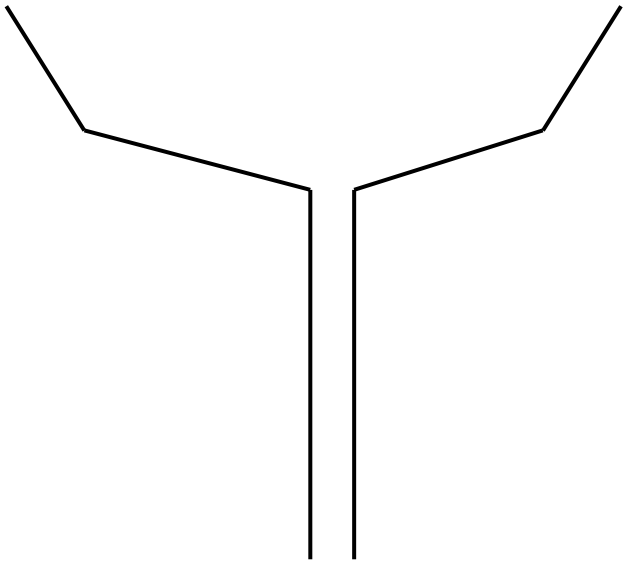
==


24
76
47
93
32
11
Is this 47? (1)
Is this 47? (3)
Is this 47? (2)
Is this 47? (4)

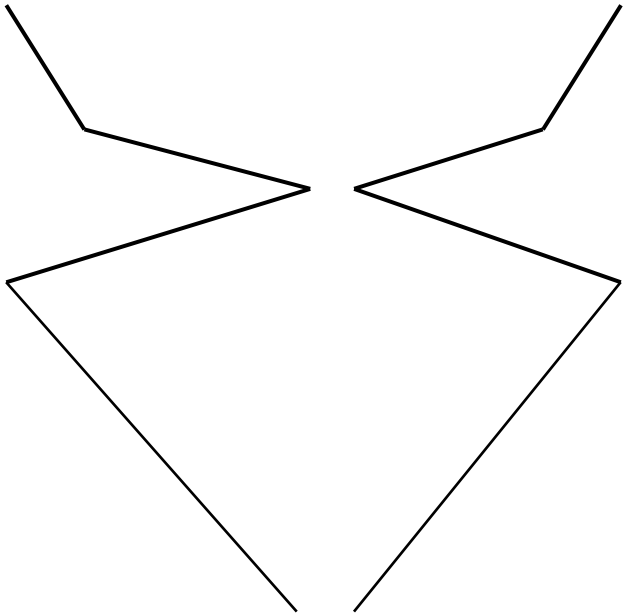

A
B
C
D
F
G
H

Random Walks are Inefficient

- Particularly in the case of troubleshooting
  - You have a scoped space
  - You know something exists that you need to find
  - But it could be something that is missing, rather than something that is there
- And yet…
  - The guided random walk is the most common troubleshooting technique deployed

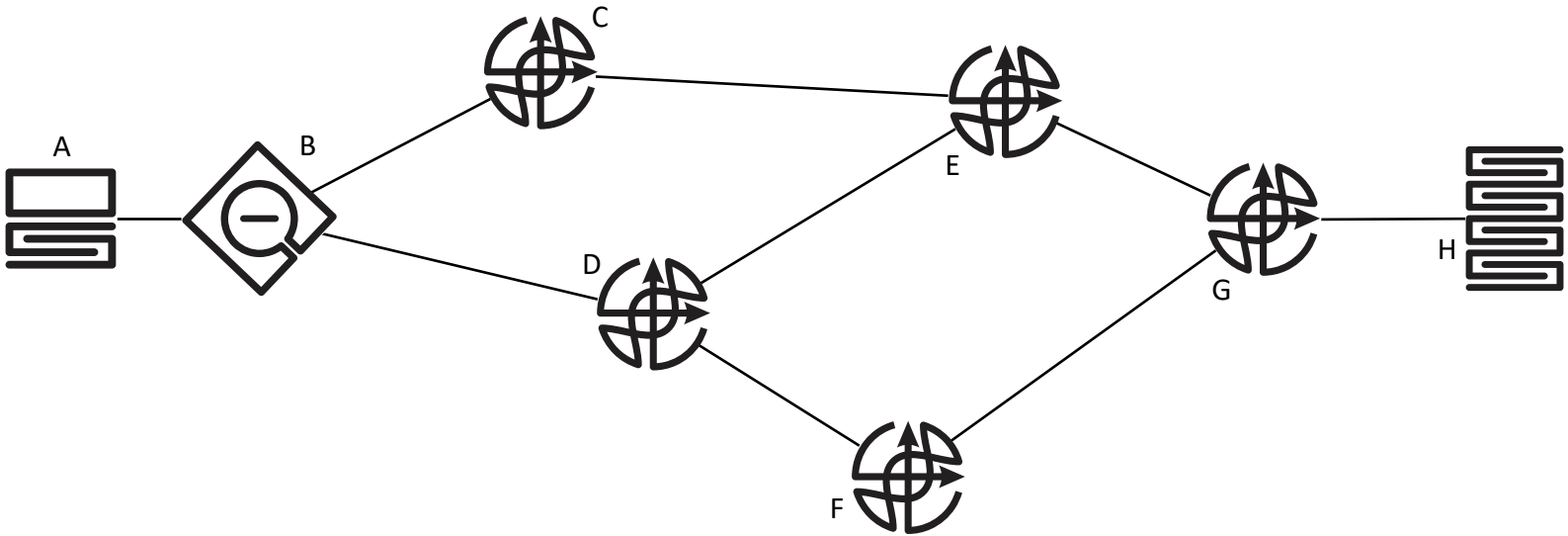


“I’ve seen this before, and I know the solution”



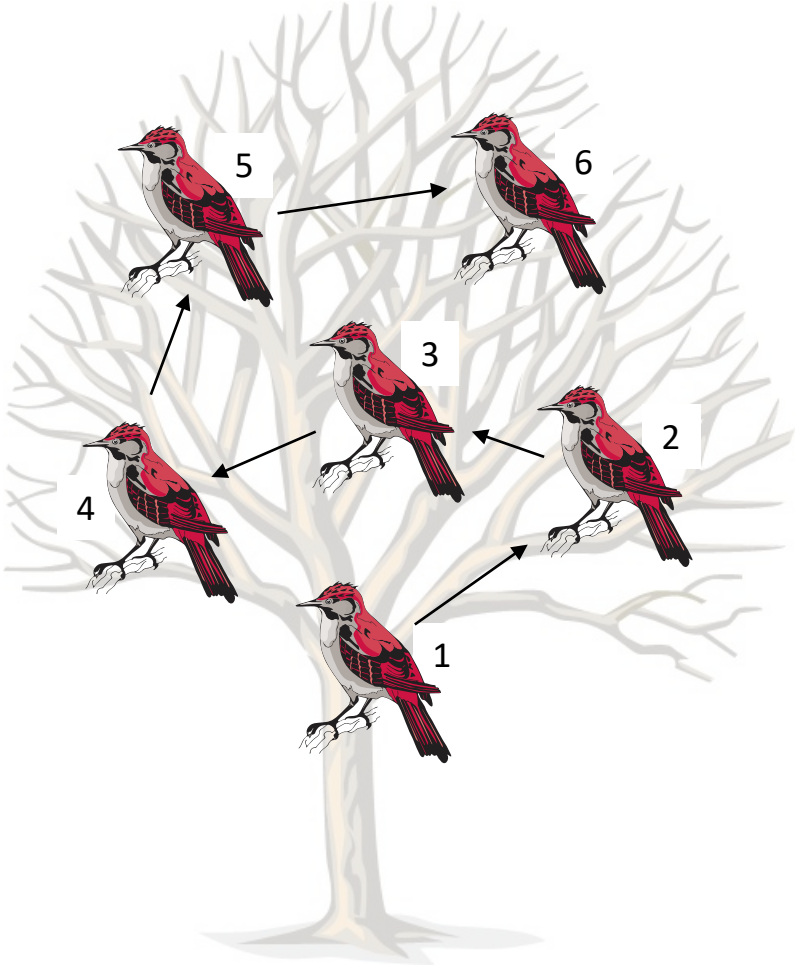
“I thought I knew the problem, but it turns out I was wrong”

Go back to basics!

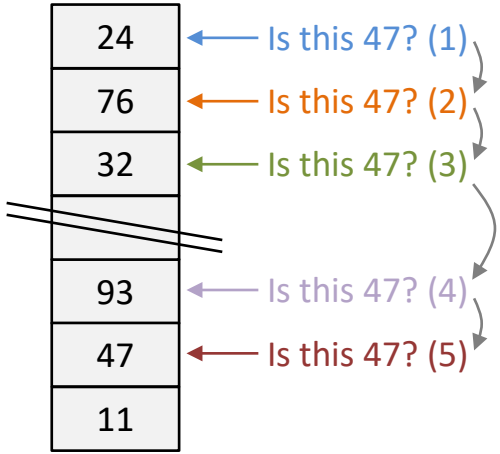

A
B
C
D
E
F
G
H

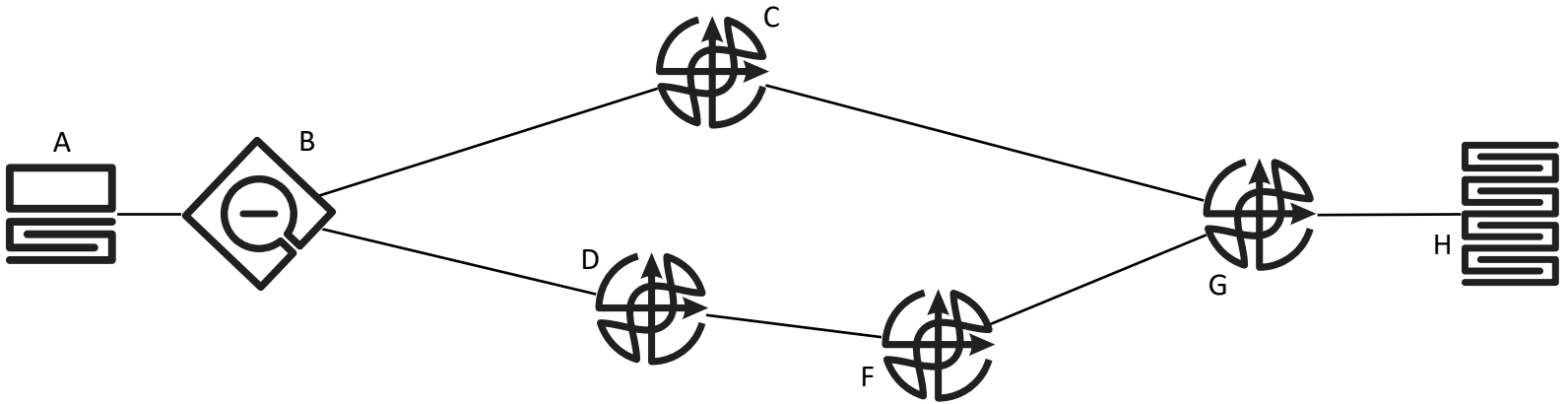
Avoiding “the Narrows”

- Each of these…
  - If asked before you have narrowed to that point in the network…
  - Will take you down the narrows
- Keep your focus on
  - What the system is supposed to do
  - What the system is doing
- Rather than guessing about what might be wrong
  - Or what you can measure


5
6
3
2
4
1

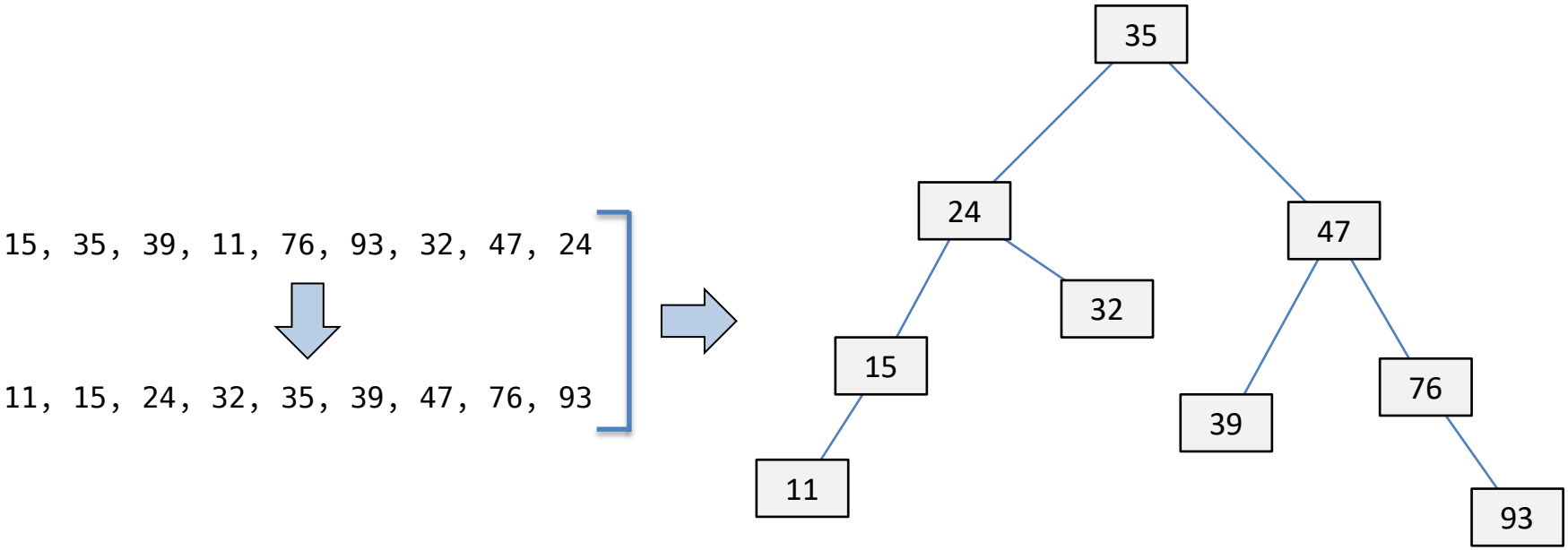
==

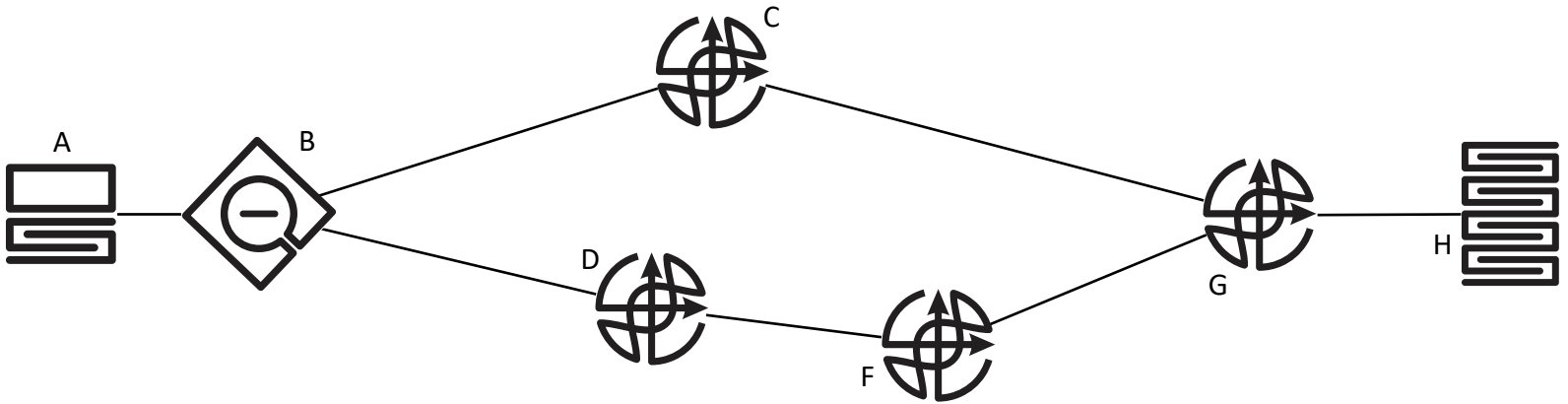

24
76
32
93
47
11
Is this 47? (1)
Is this 47? (2)
Is this 47? (3)
Is this 47? (4)
Is this 47? (5)

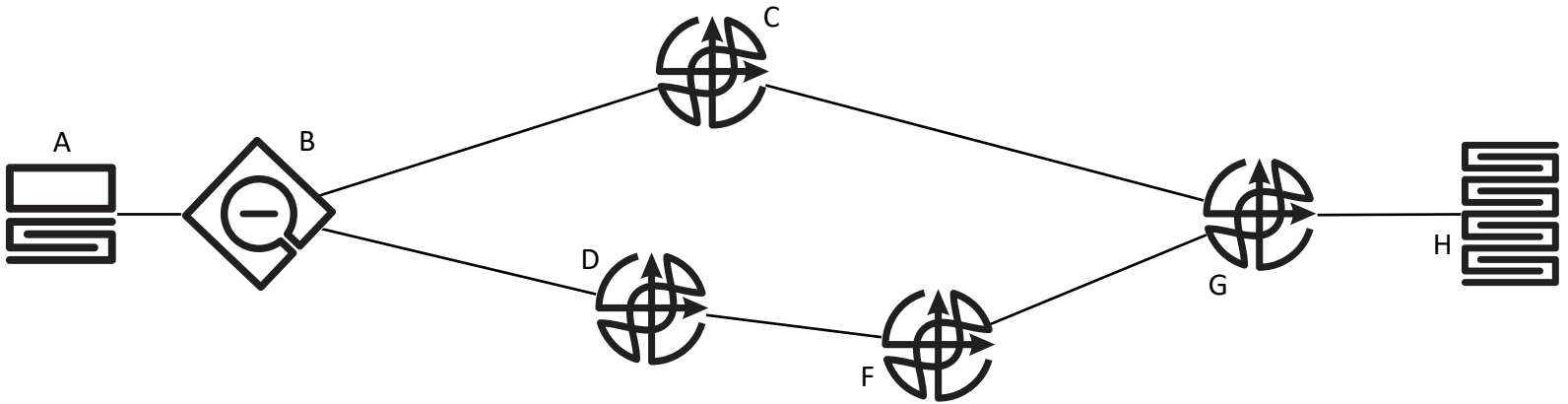

A
B
C
D
F
G
H

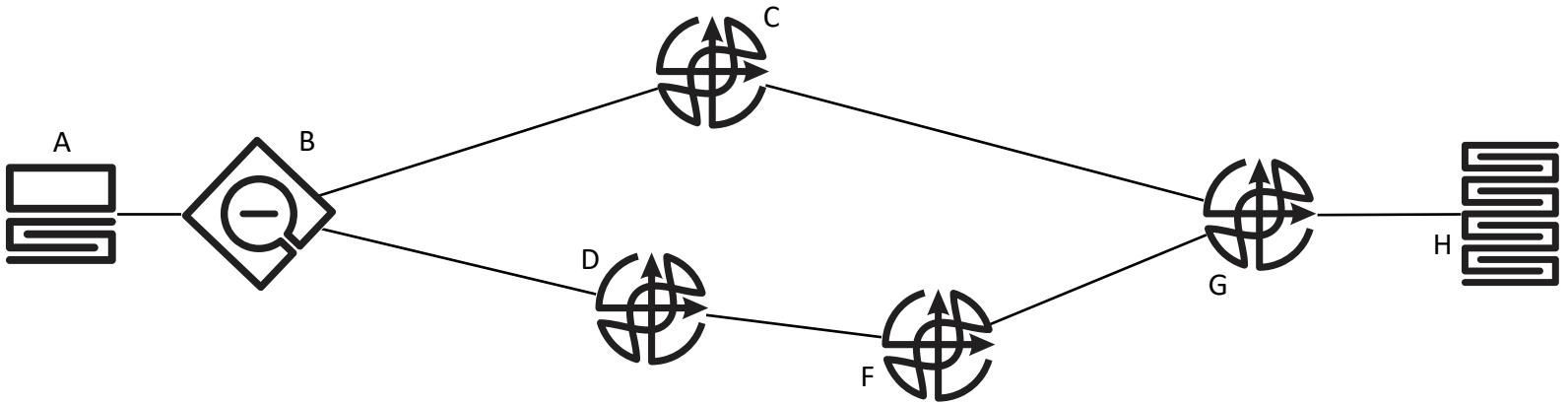
Linear Process

- More methodical
- Could be useful in some situations
- But not likely to find the problem quickly in most networks…


15, 35, 39, 11, 76, 93, 32, 47, 24
11, 15, 24, 32, 35, 39, 47, 76, 93
35
24
47
32
15
39
76
11
93

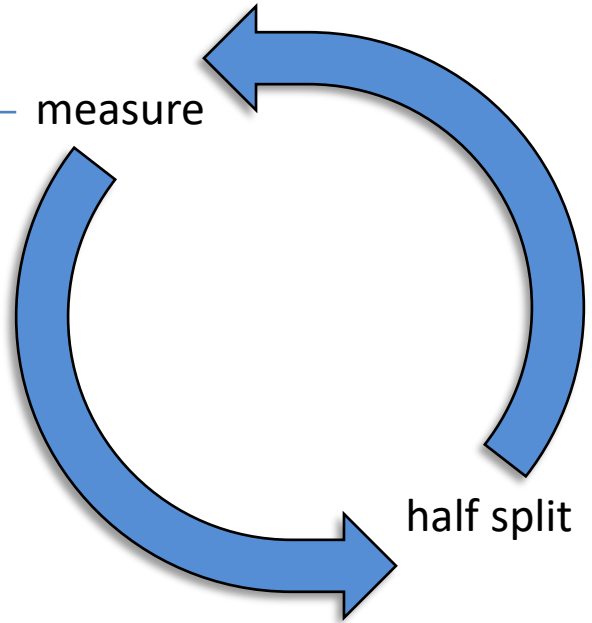

A
B
C
D
F
G
H


A
B
C
D
F
G
H


A
B
C
D
F
G
H

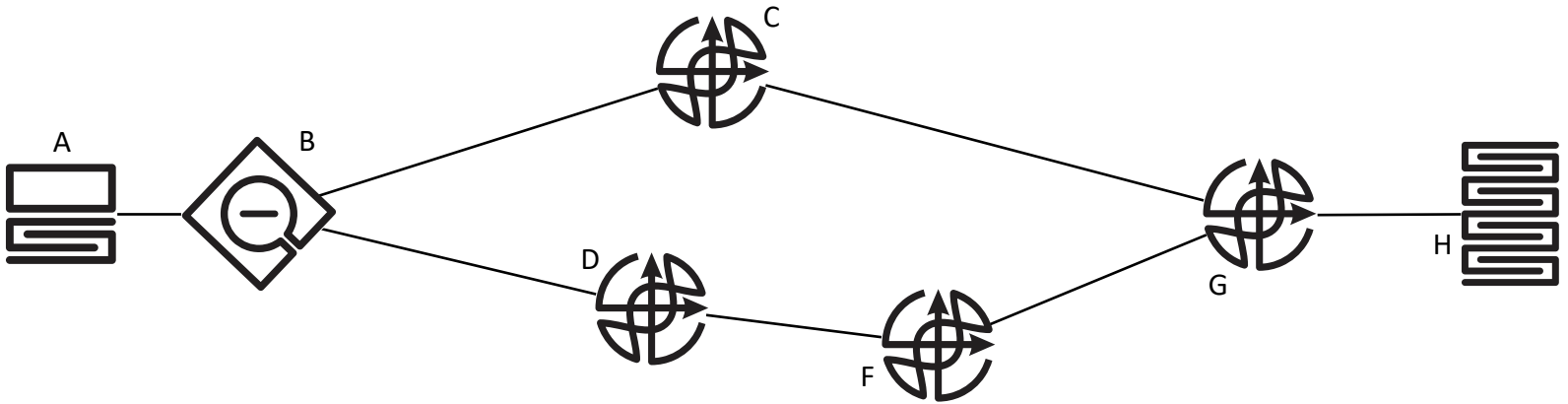
Putting it All Together:

A Troubleshooting Process

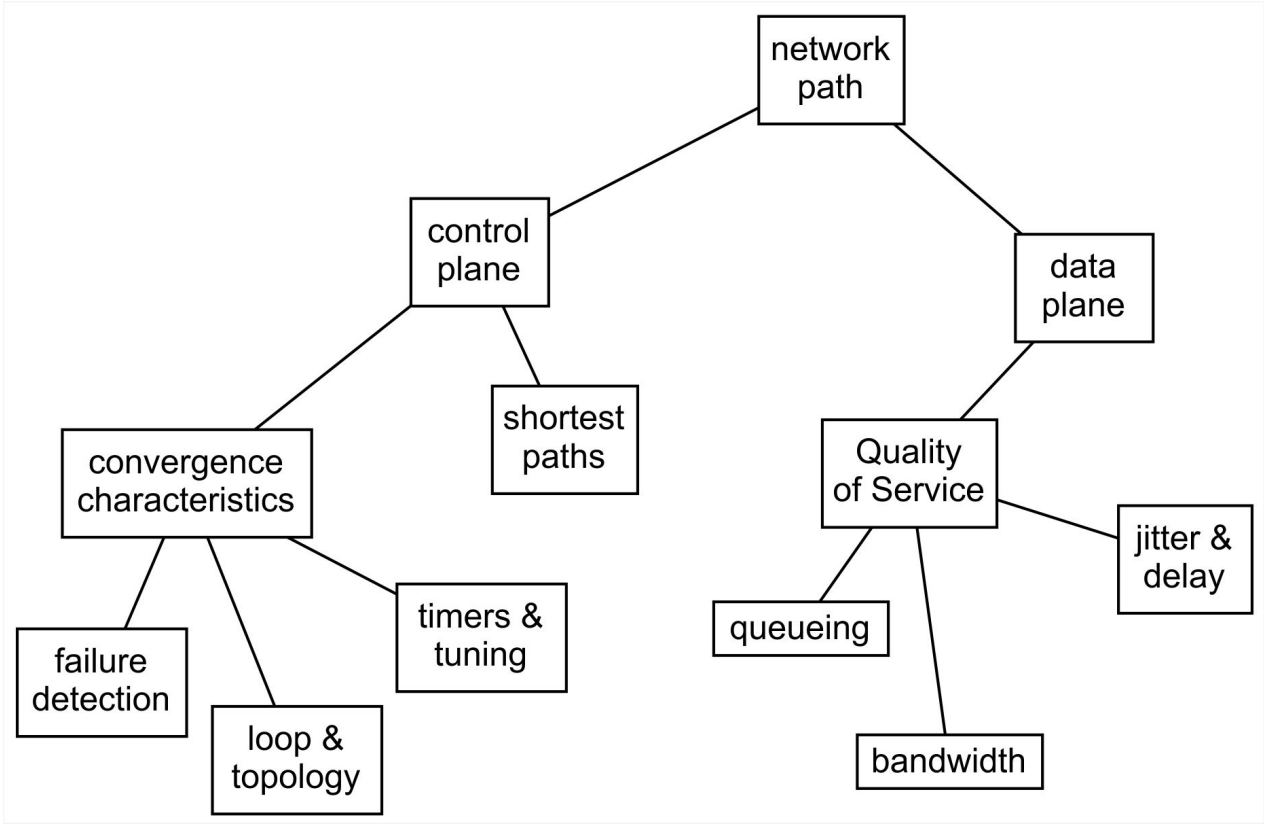


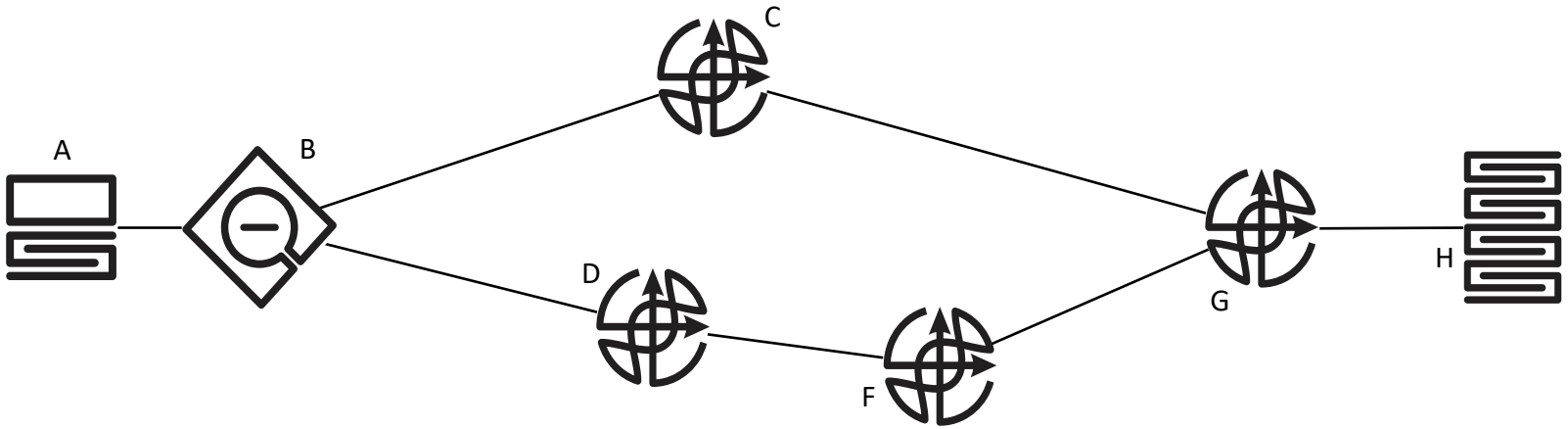
observe/orient
what should I measure?
how can I measure it?
how might measuring change things?
what should normal look like?
why might it look like this?

decide/act
should I go left or right?
should I go in or out?


A
B
C
D
F
G
H

Application
Network Stack
Output Queue
PHY
PHY
Input Queue
Policy/Switch
Output Queue
PHY
PHY
Input Queue
Switch
Fabric
Output Queue
PHY
PHY
Input Queue
Switch
Output Queue
PHY
Application
Network Stack
Output Queue
PHY
A
B
C
D
F
G
H


network path
control plane
data plane
shortest paths
convergence characteristics
Quality of Service
jitter & delay
timers & tuning
failure detection
loop & topology
queueing
bandwidth

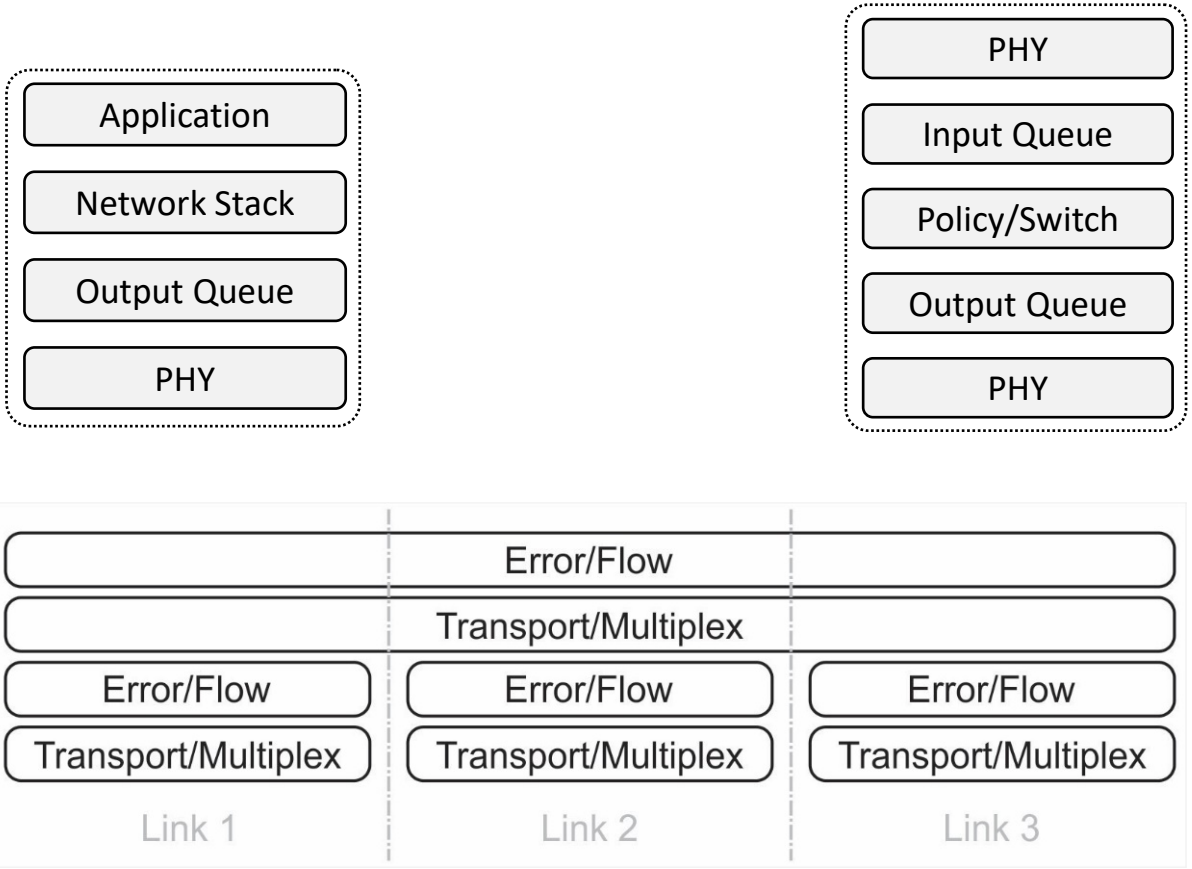

A
B
C
D
F
G
H

Discover the Background

- Ask WHAT
  - What do I need to measure?
- To do this I must
  - Understand how this works
  - Understand what I should expect to see
  - Having a solid mental model of the system is going to uncover these things
  - Where are there interfaces between systems?

Discover the Background

- Ask HOW
  - How can I measure this?
- To do this I must
  - Understand how I can discover whether are not my expectations are correct
  - Understand how I can discover if what I think should be there *is really there*
  - Understand *where* I can measure this effectively


Application
Network Stack
Output Queue
PHY
PHY
Input Queue
Policy/Switch
Output Queue
PHY
Error/Flow
Transport/Multiplex
Error/Flow
Transport/Multiplex
Error/Flow
Transport/Multiplex
Error/Flow
Transport/Multiplex
Link 1
Link 2
Link 3

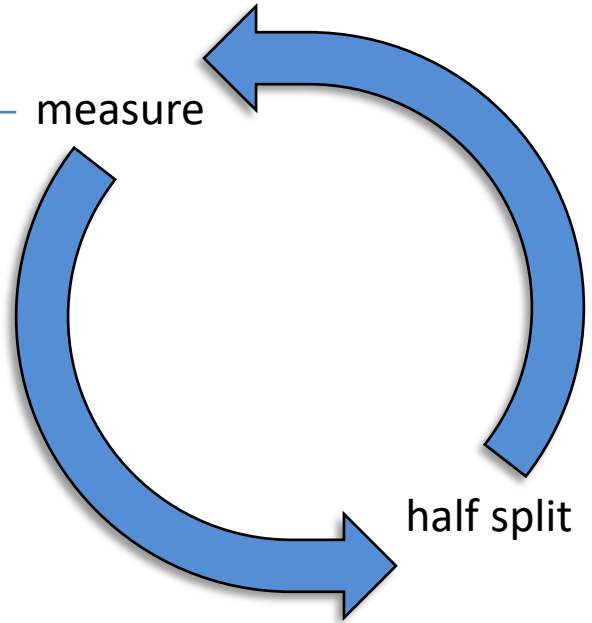
Build lifelines into adjacent systems *before the failure*

Summary

- Building understanding before you encounter a failure will help you work faster and avoid the narrows
- Build a baseline of what is normal
  - Gather information from your measurement points periodically
  - Know what things “should” look like
- Model the systems mentally
  - How does this work?
  - Why does this work this way?

Summary

- Build decision points into the system
- Think about where the half split points are
  - Design your network with choke points to make half splitting easier
- Think about how you can manipulate things
  - To understand what is going on
  - Without causing damage to adjacent systems

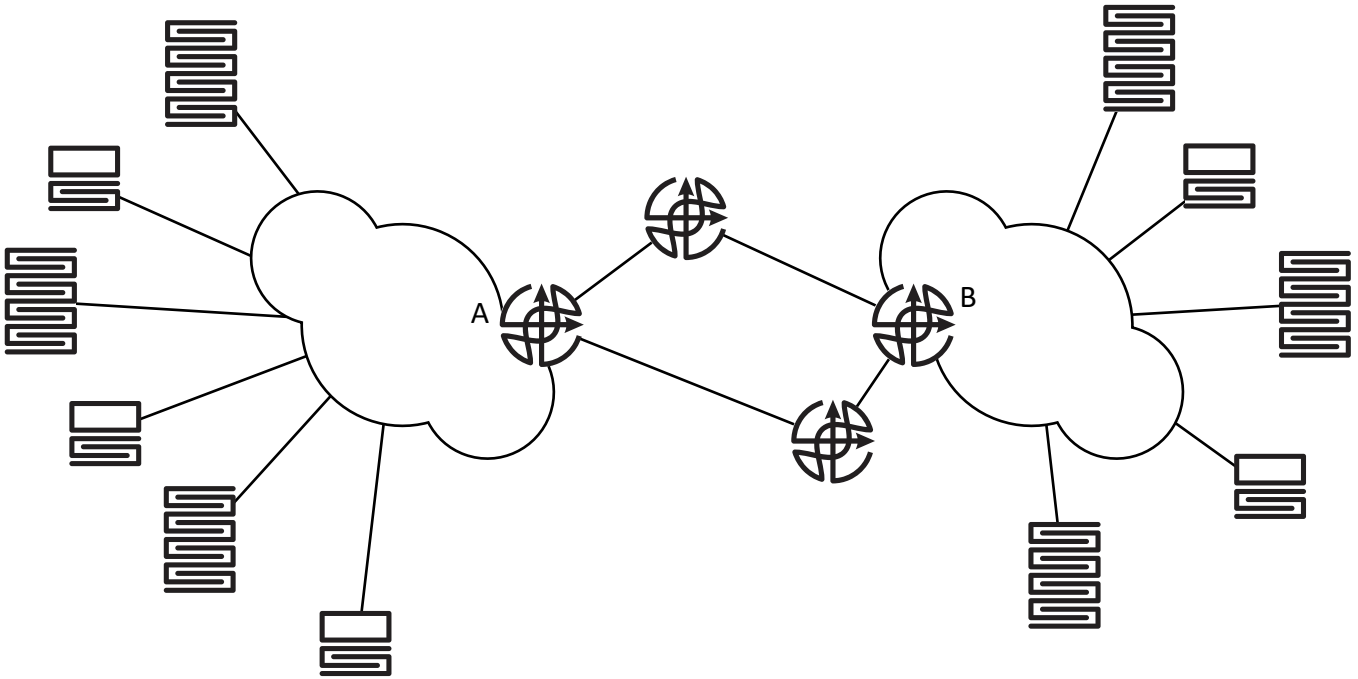


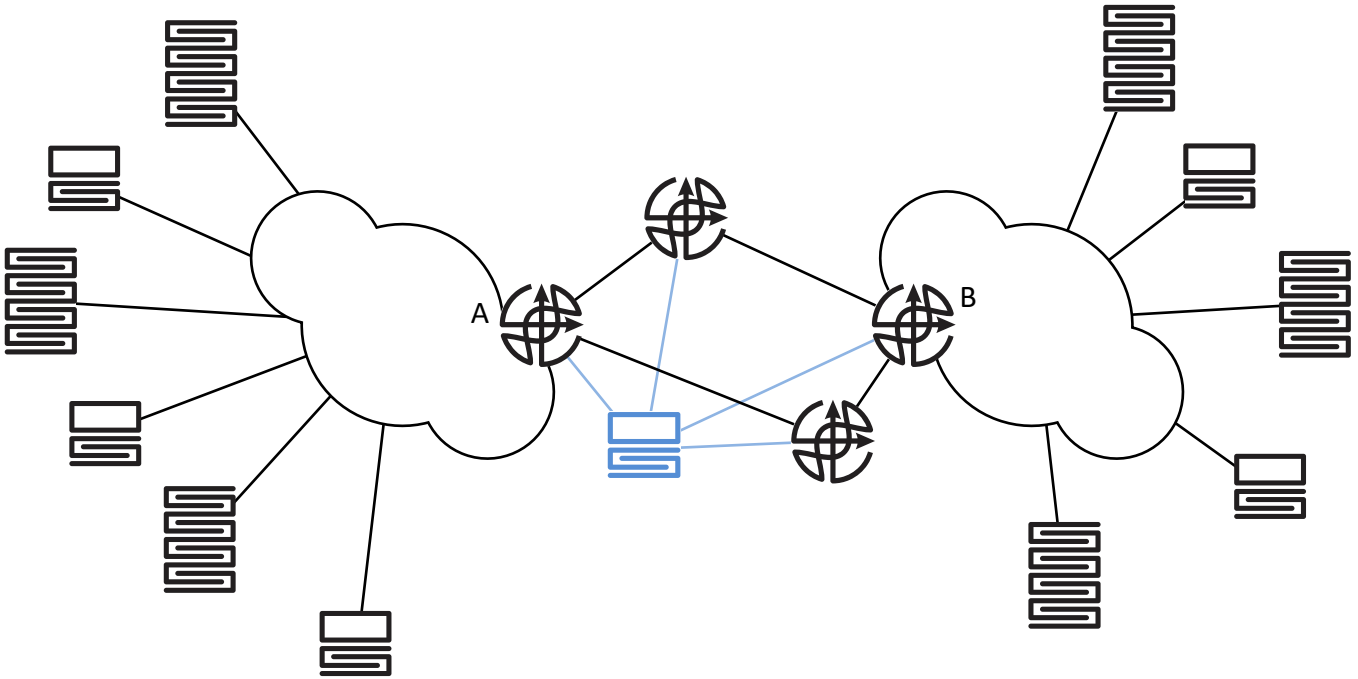
observe/orient
what should I measure?
how can I measure it?
how might measuring change things?
what should normal look like?
why might it look like this?

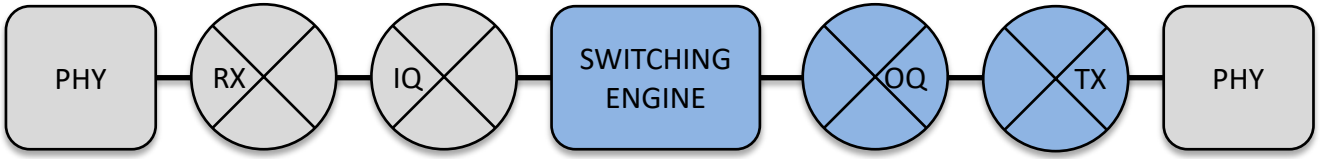
decide/act
should I go left or right?
should I go in or out?

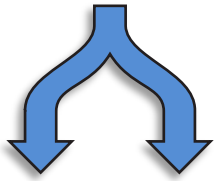
Example

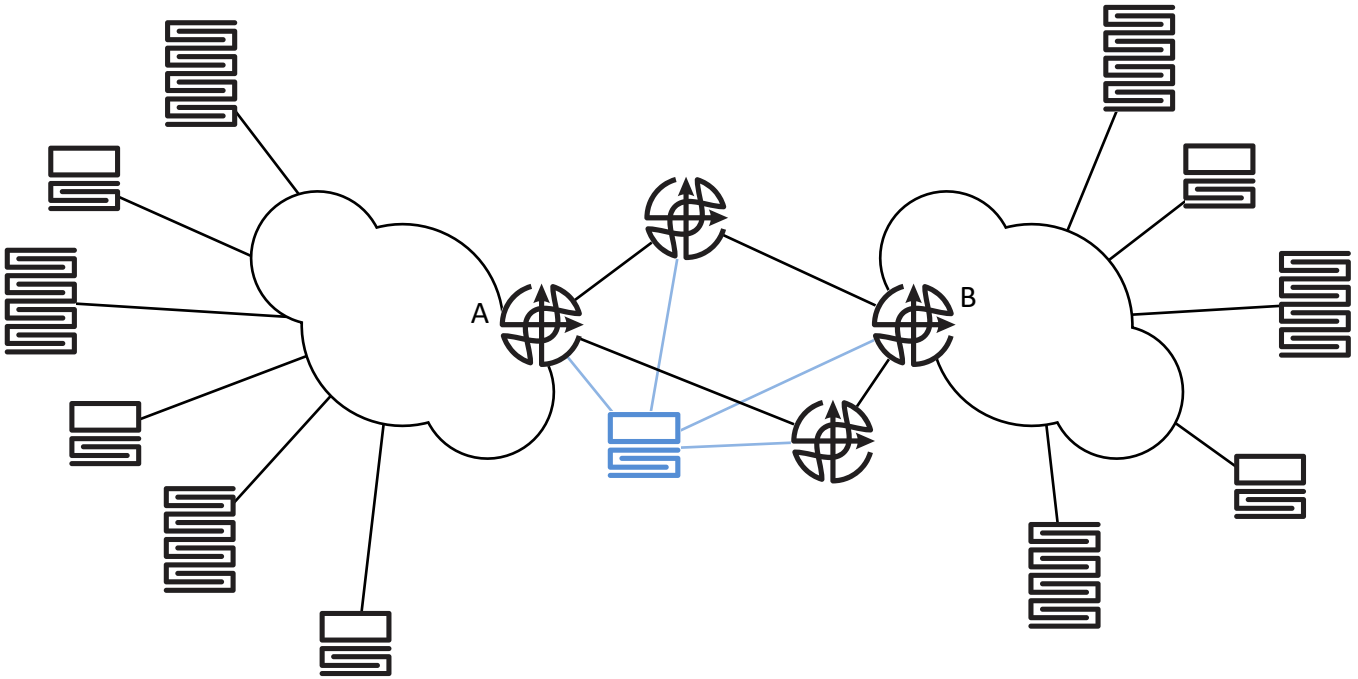
Flapping EIGRP

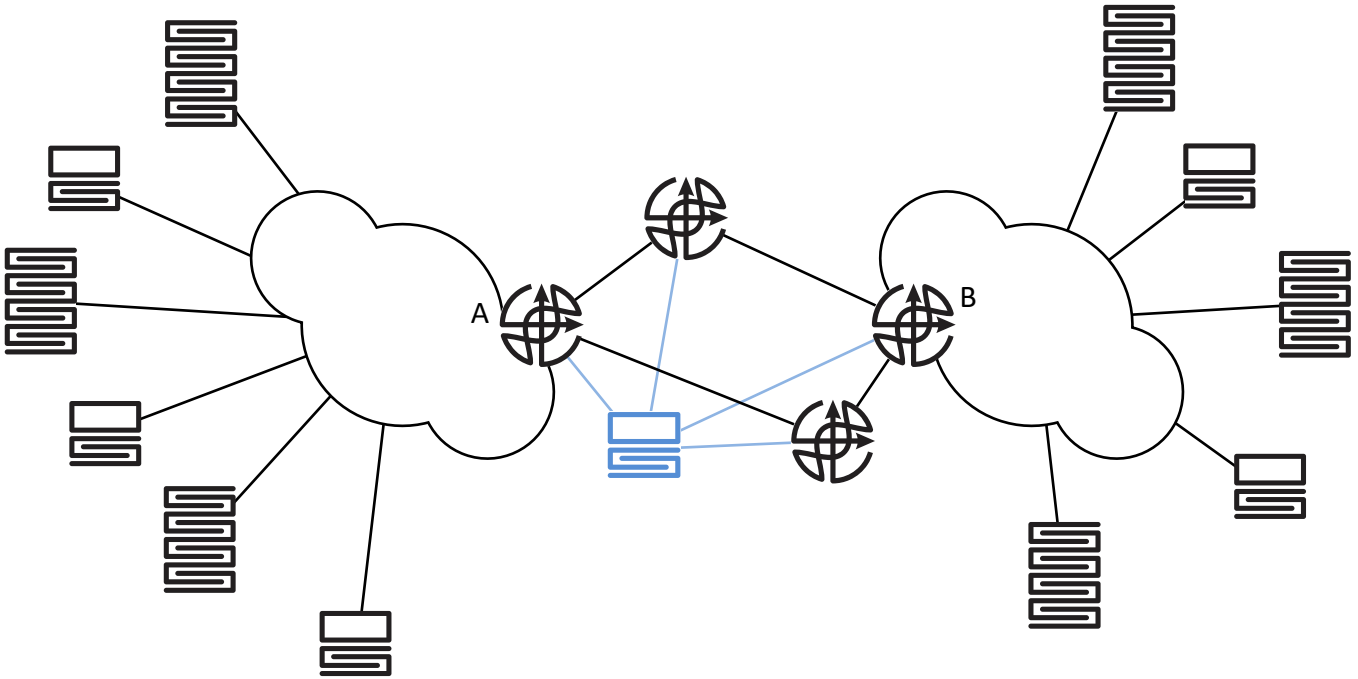

A
B

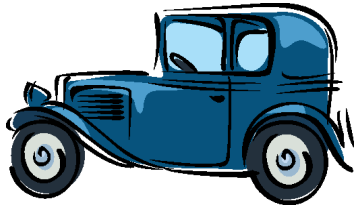

A
B

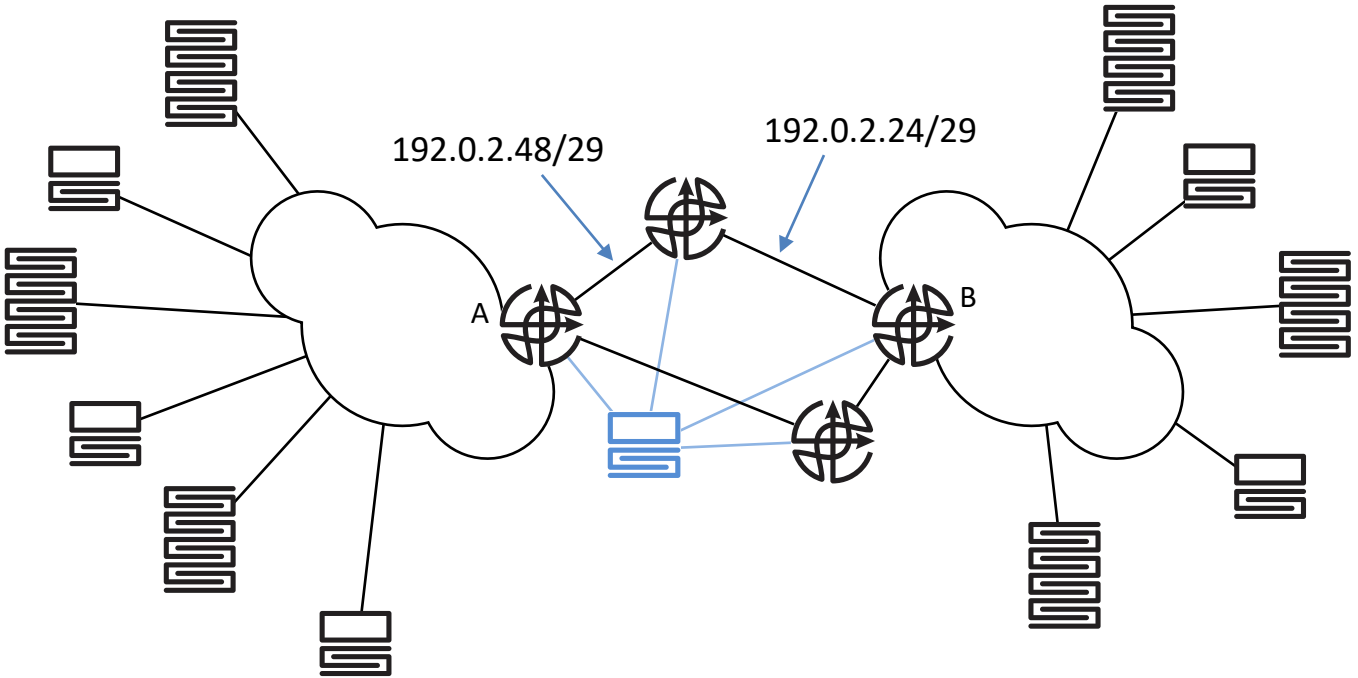

PHY
RX
IQ
SWITCHING ENGINE
OQ
TX
PHY


network
router


A
B


A
B

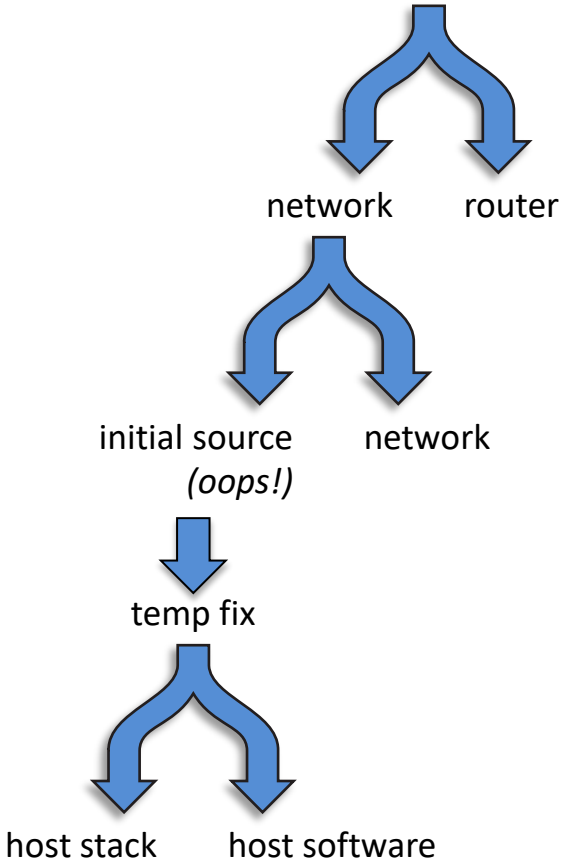



192.0.2.48/29
192.0.2.24/29
A
B

Trace Information

- Packets are originating from every conceivable source
  - Not just network management hosts
  - So this is not likely a network management application
- We can't measure every host in the network …
  - Choose a small set to measure

192.0.2.7
192.0.2.15
192.0.2.23
192.0.2.31
. . .


network
router
initial source
(oops!)
network
temp fix
host stack
host software

Host Troubleshooting

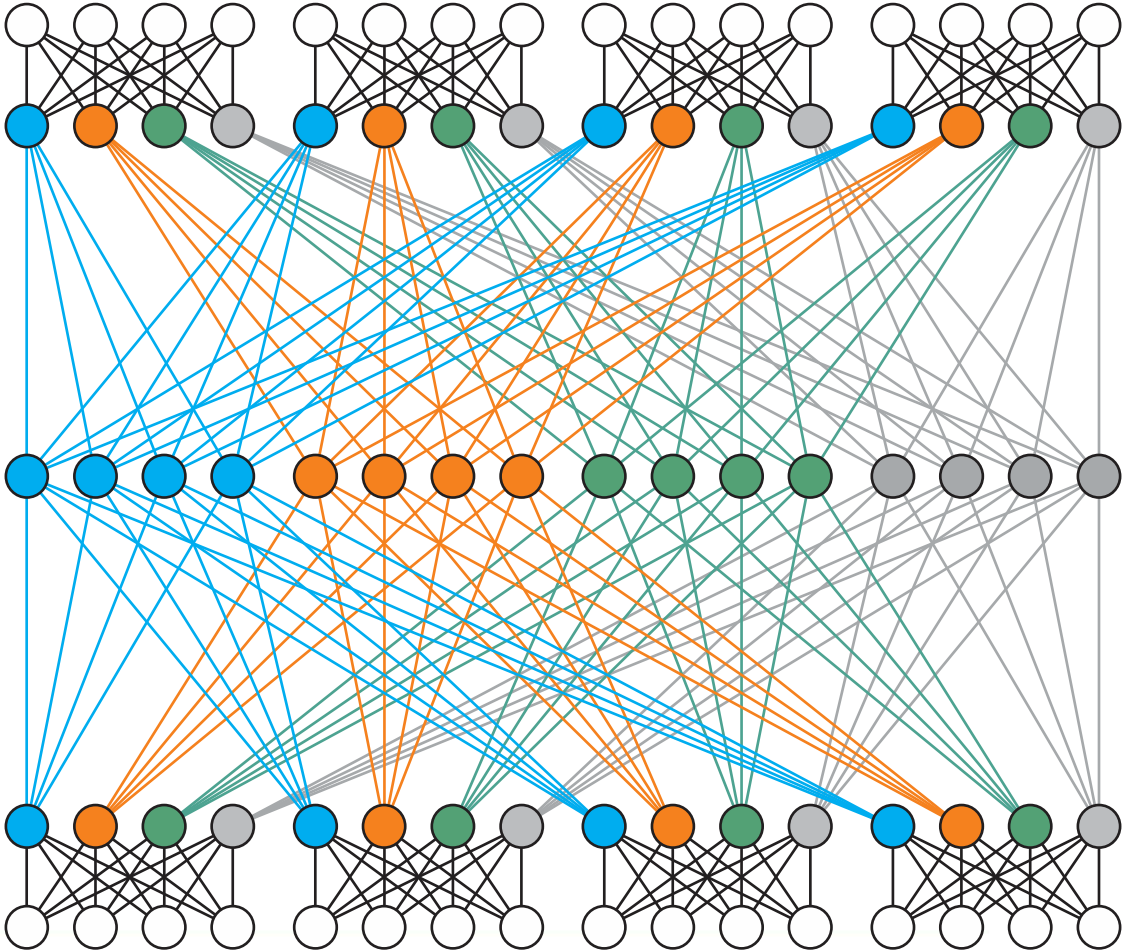
- What has been installed recently?
  - Ask the obvious question first!
- Measure
  - Disable this software on two hosts machines
  - Leave the software active on two other hosts
  - Note their IP addresses
  - Log IP packets at a core router with these four source addresses
- Result
  - No problematic subnet broadcasts being transmitted by the two modified hosts
- Root cause!

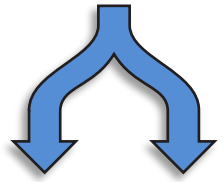
Root Cause

- Software installed on all machines was intended as the *server half* of a backup package
- To discover clients
  - Sends subnet broadcasts
  - To *every possible* subnet on classful boundaries
- Installed on more than 10k machines
  - Subnet broadcasts being replicated by *every* router in the network
  - Filling input queues
  - EIGRP was just the “canary in the coal mine”

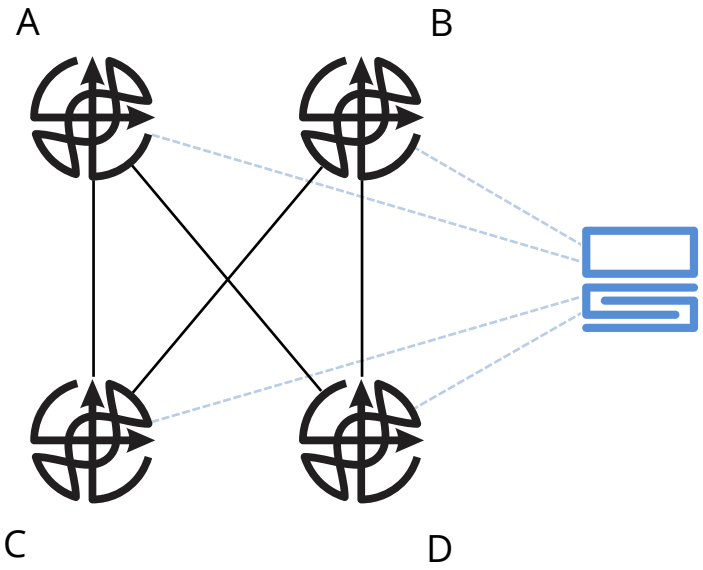
Example

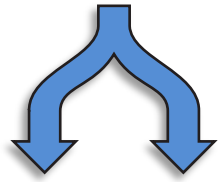
Flapping BGP



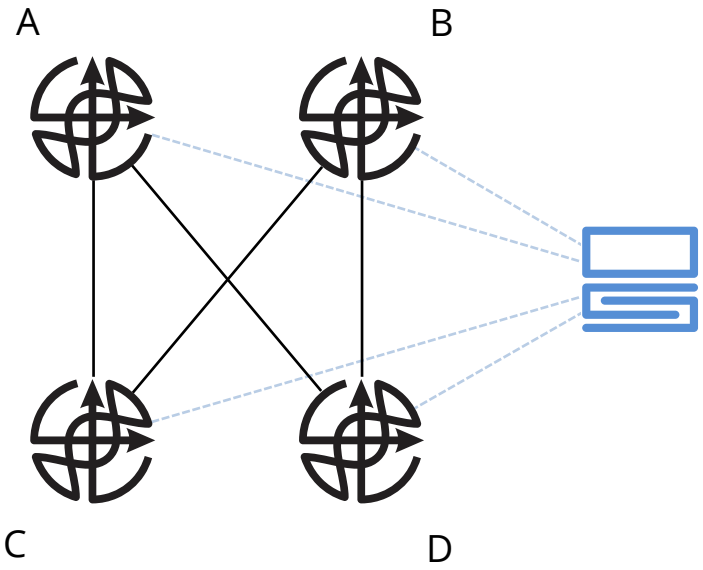


bgp configuration link failures


A
B
C
D

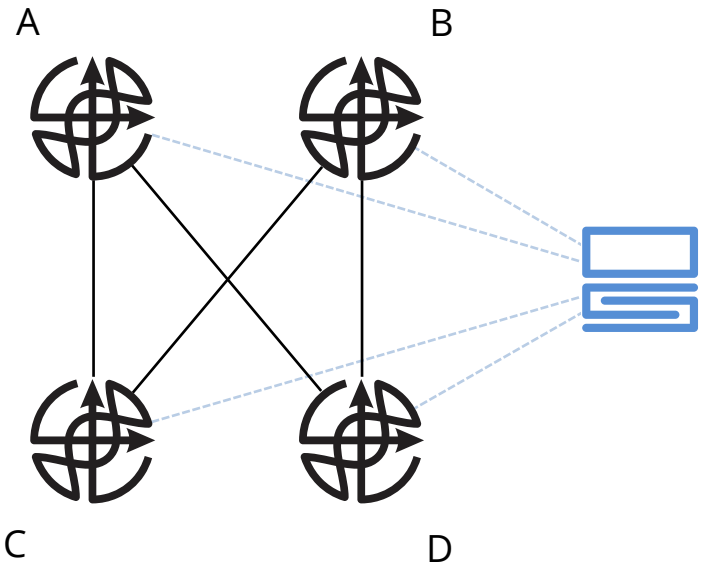


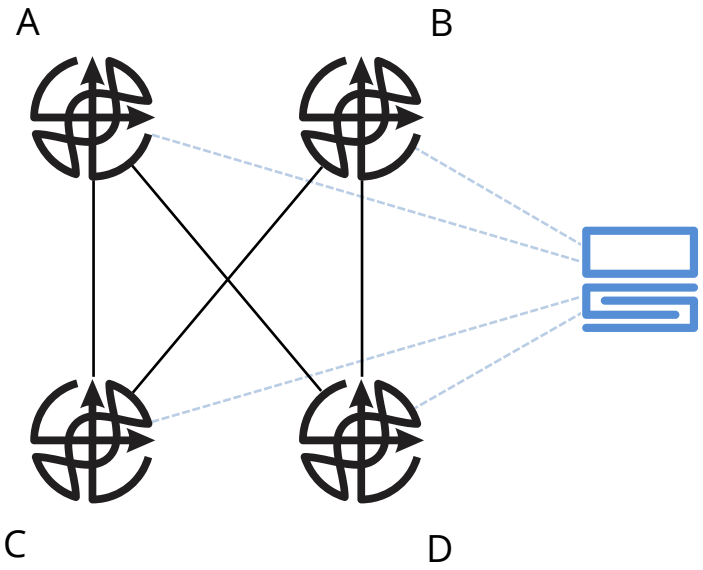
bgp configuration link failures

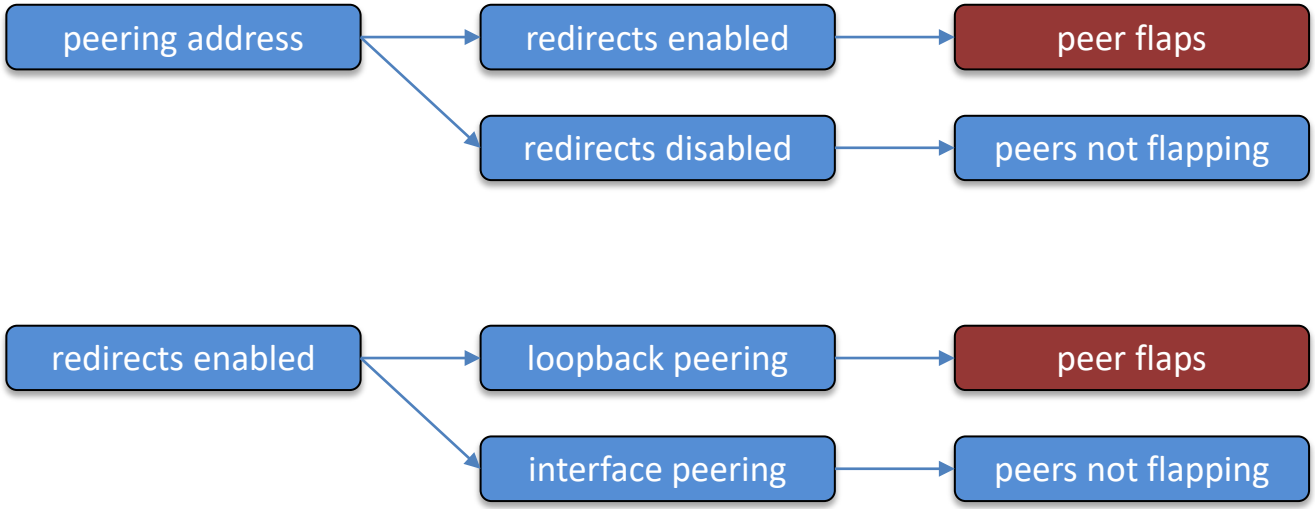

A
B
C
D

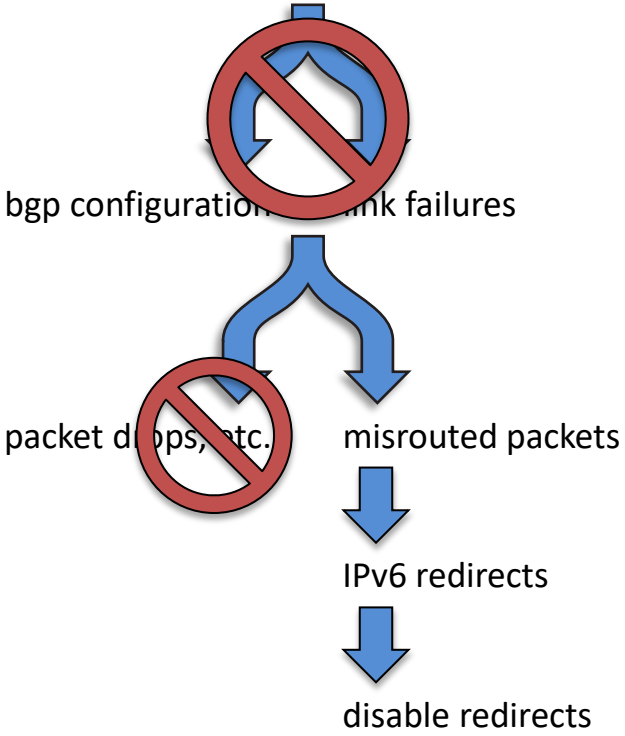
bgp configuration ink failures

packet drops, etc. ???


A
B
C
D

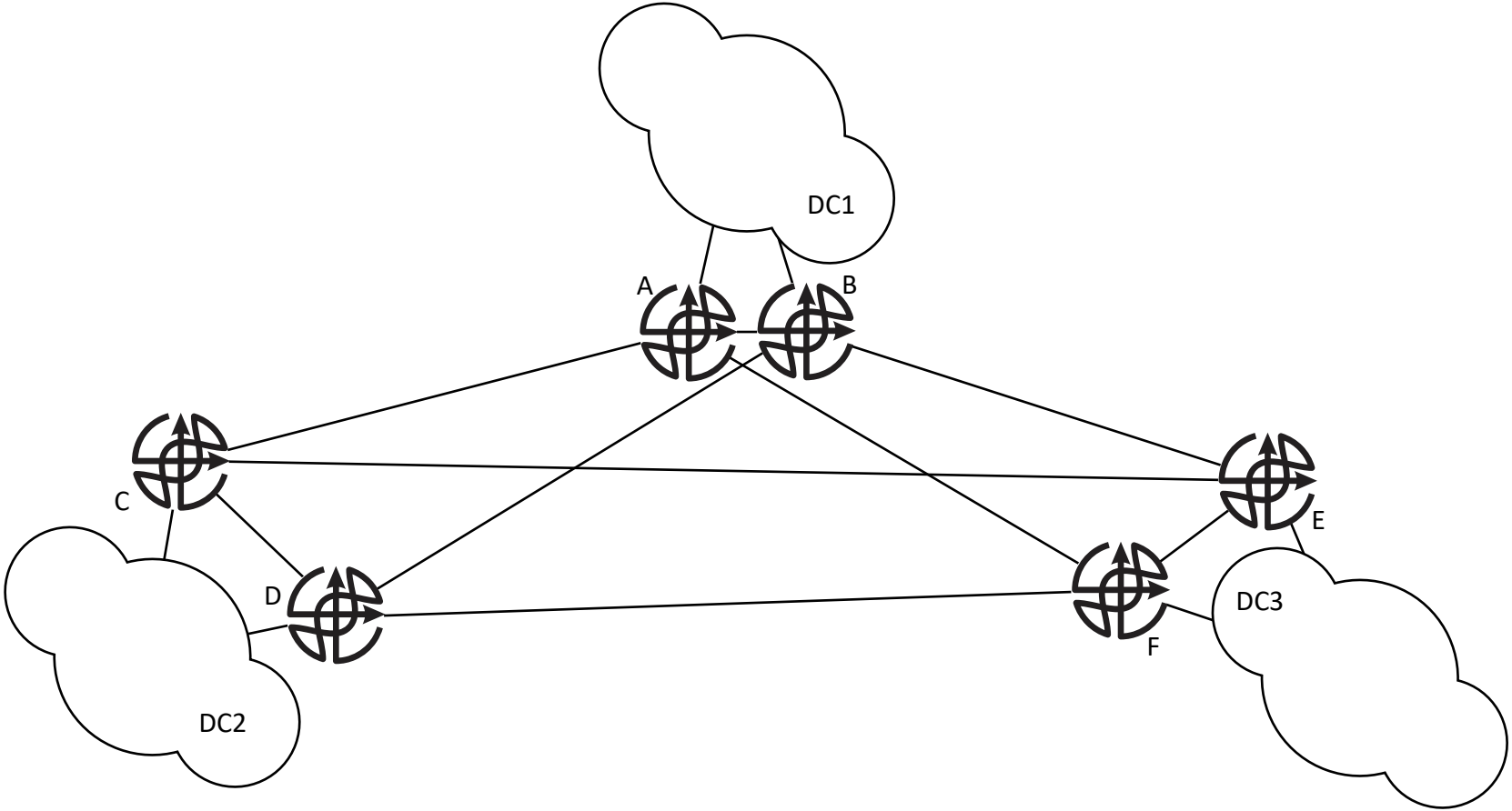

A
B
C
D

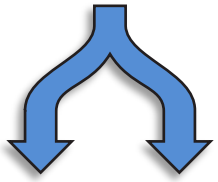

peering address
redirects enabled
peer flaps
redirects disabled
peers not flapping
redirects enabled
loopback peering
peer flaps
interface peering
peers not flapping


bgp configuration
nk failures
packet d ps, tc.
misrouted packets
IPv6 redirects
disable redirects

Example

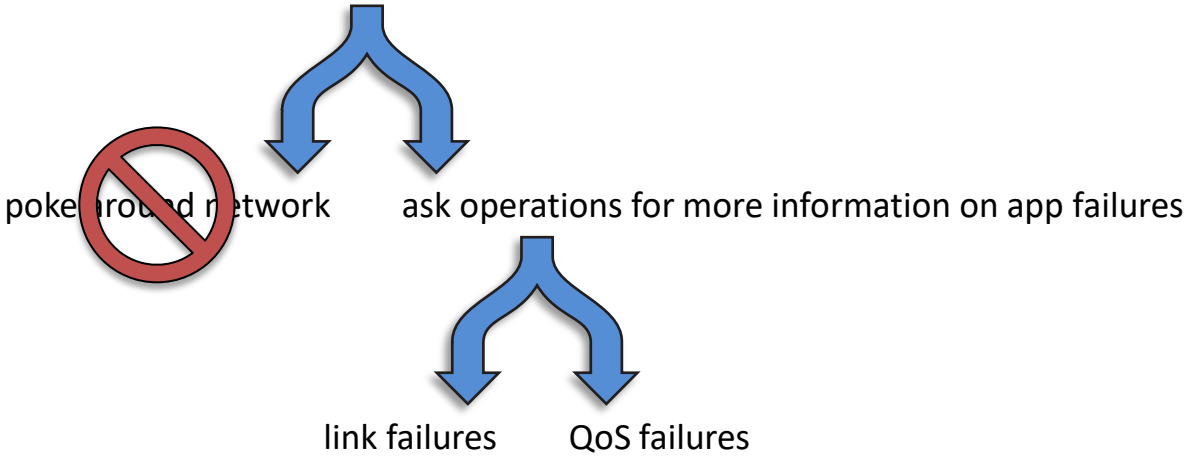
Application Performance over DCI

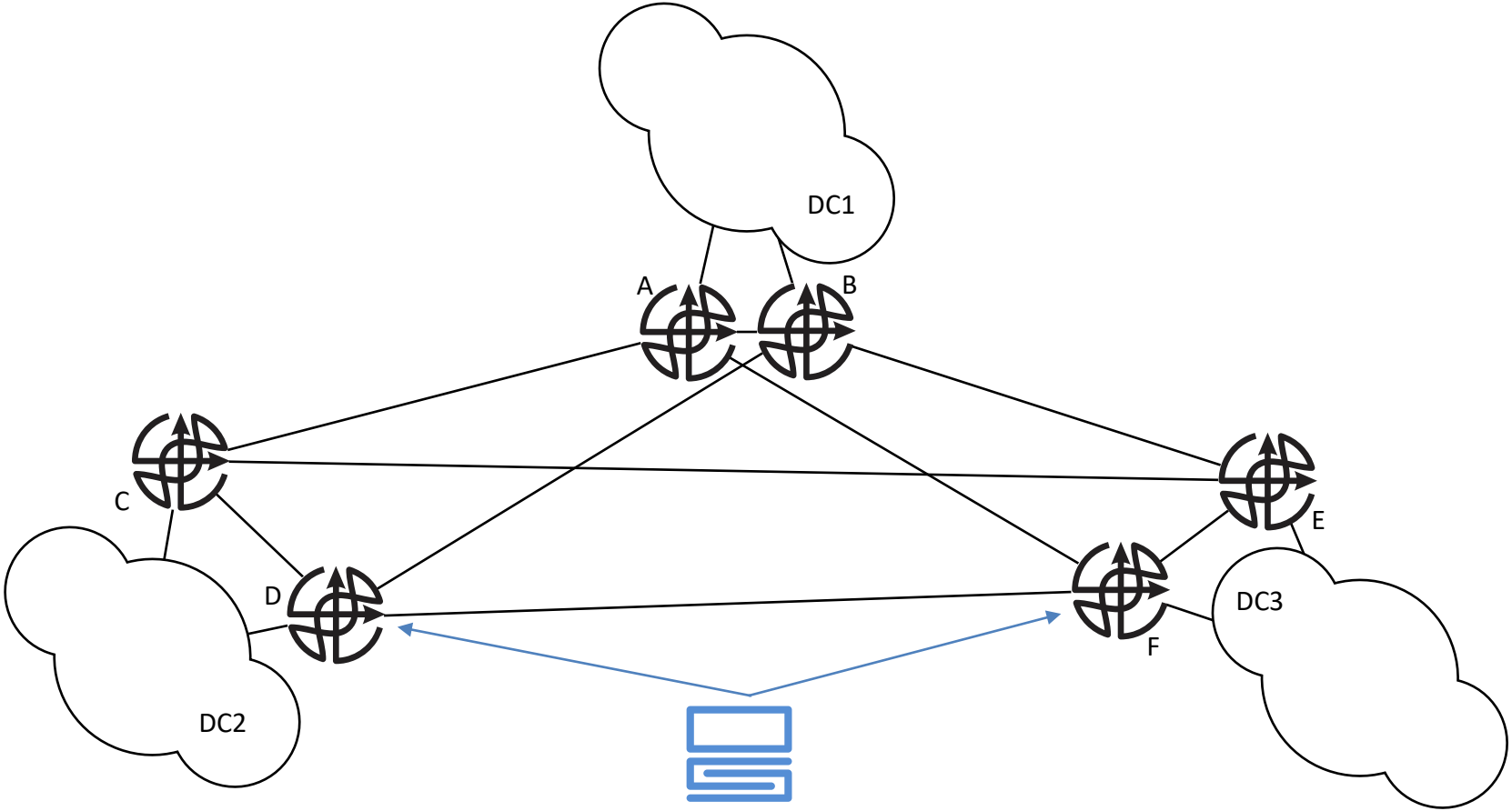

DC1
A
B
C
D
E
F
DC2
DC3

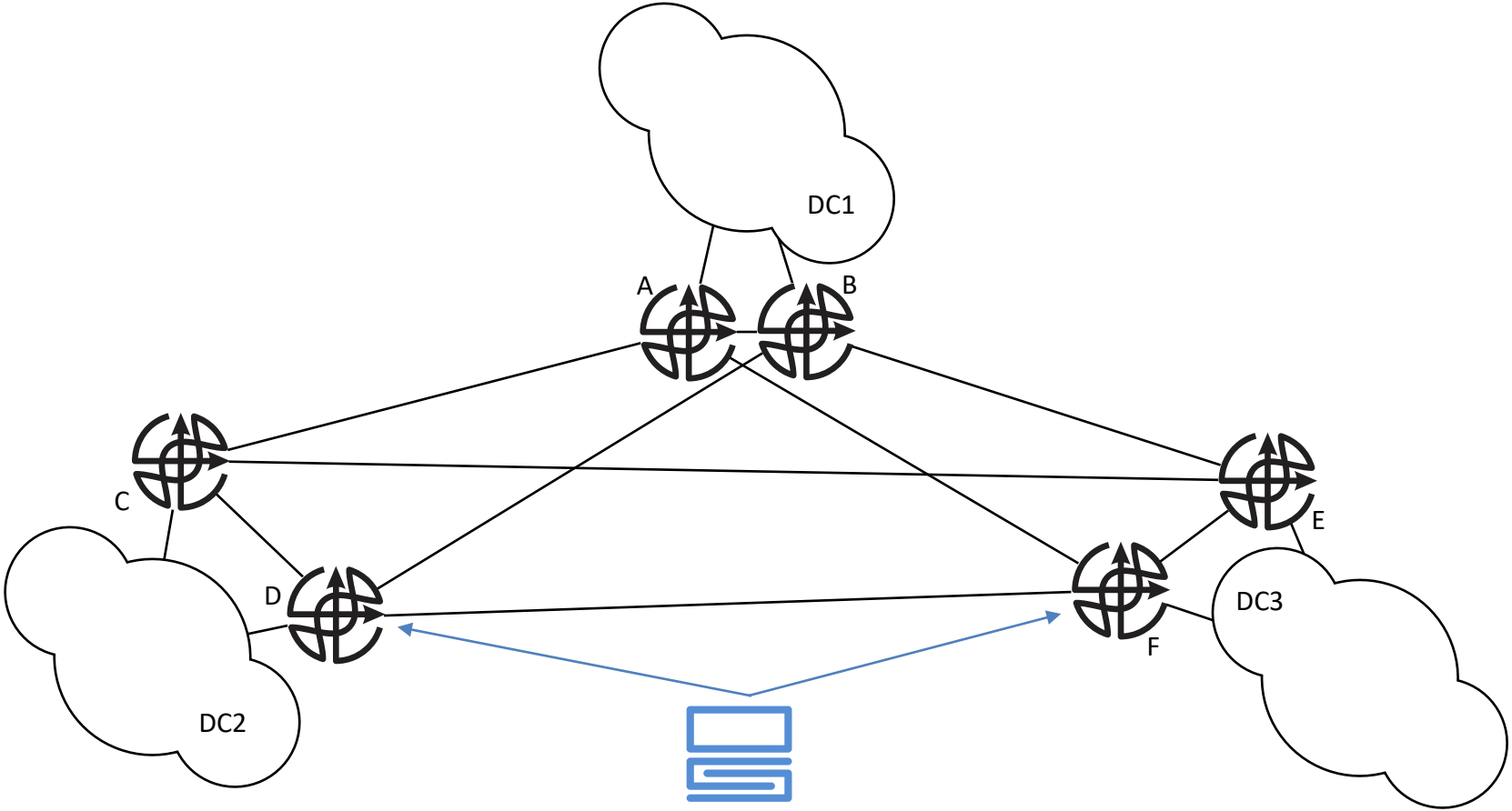


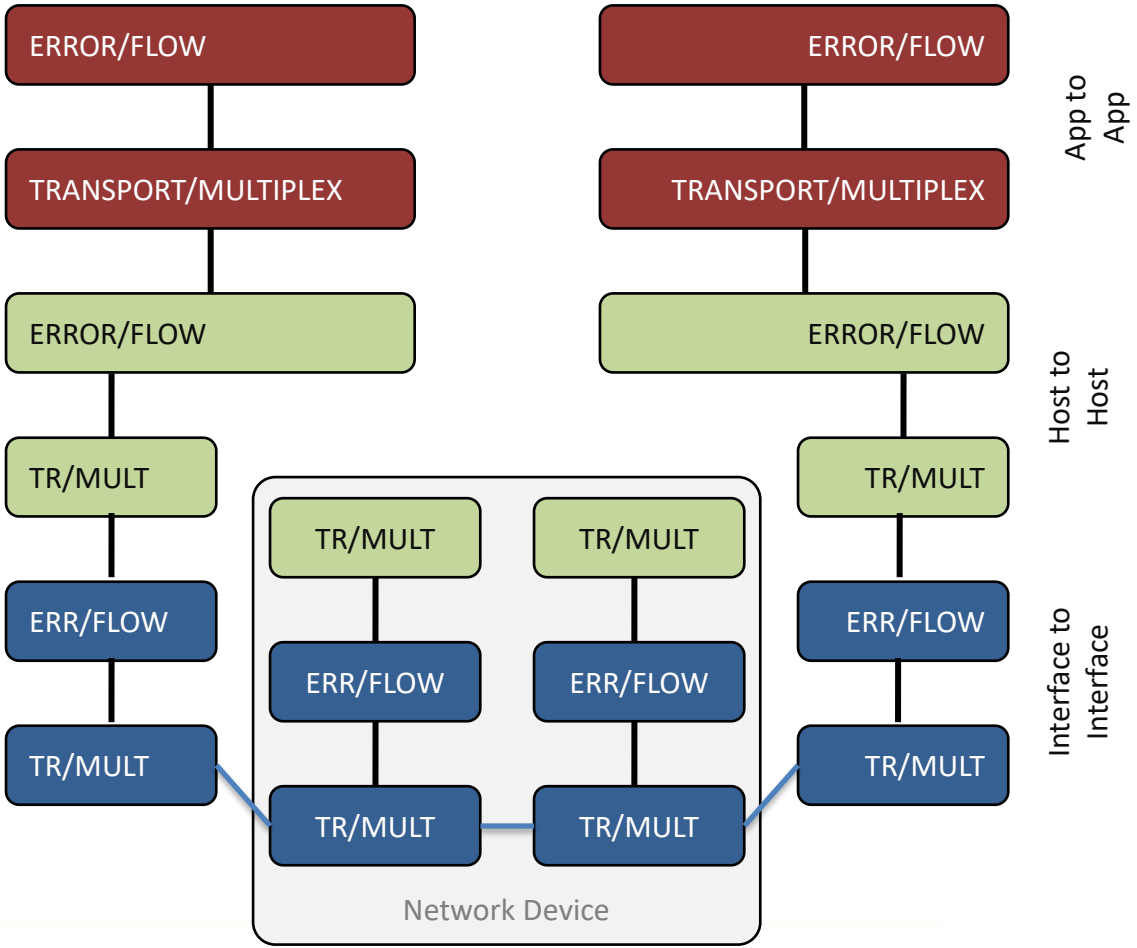
poke around network

ask operations for more information on app failures

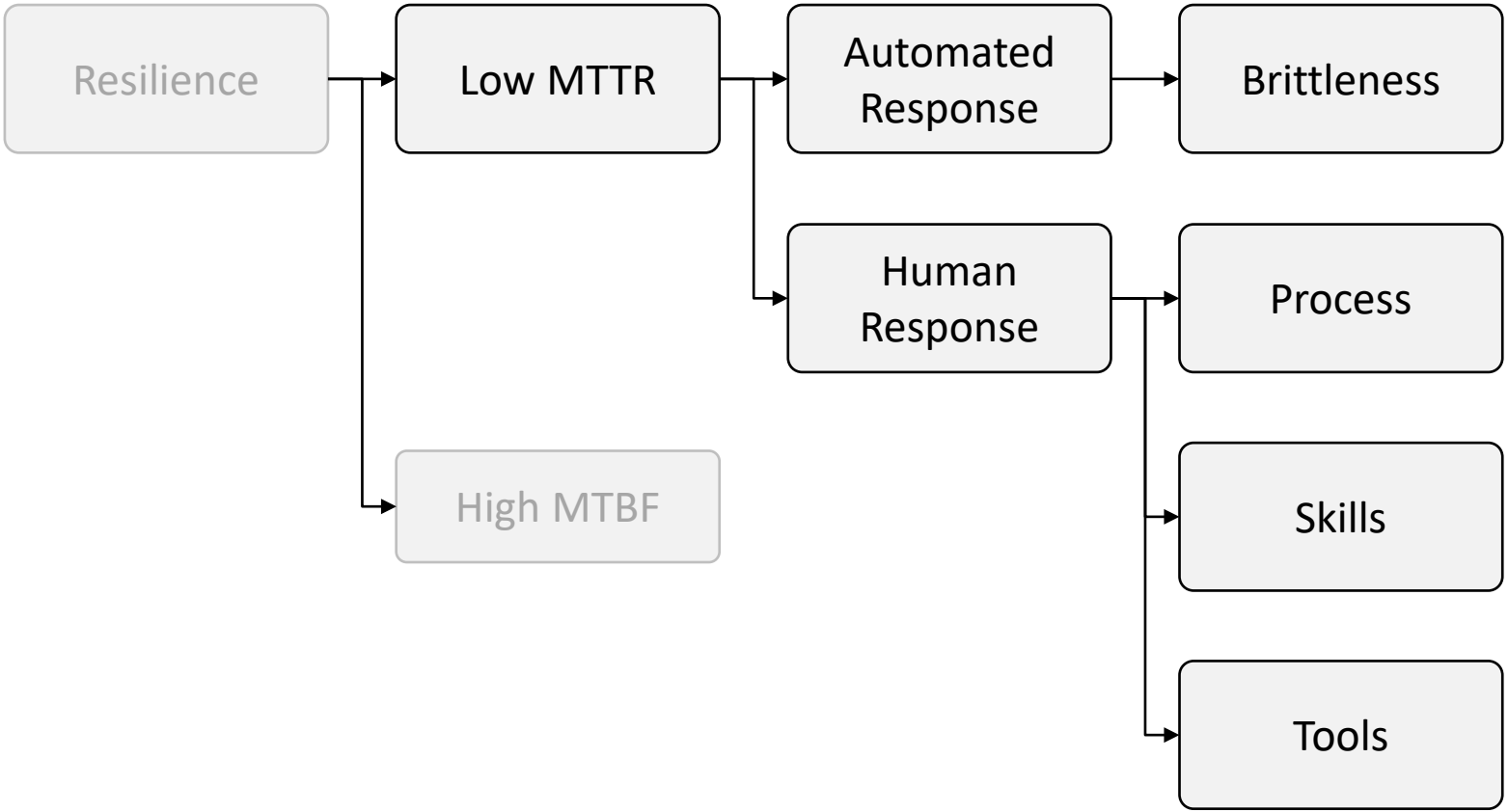

poke around network
ask operations for more information on app failures
link failures
QoS failures


DC1
A
B
C
D
E
F
DC2
DC3


DC1
A
B
C
D
E
F
DC2
DC3

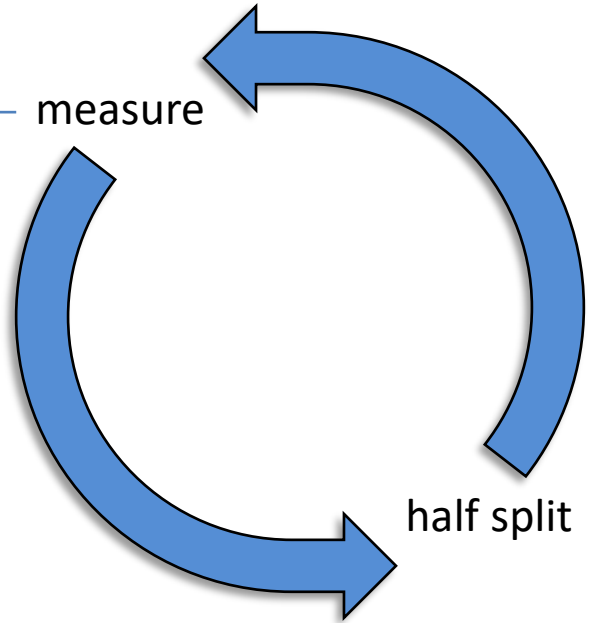

ERROR/FLOW
TRANSPORT/MULTIPLEX
ERROR/FLOW
TR/MULT
ERR/FLOW
TR/MULT
ERROR/FLOW
TRANSPORT/MULTIPLEX
ERROR/FLOW
TR/MULT
ERR/FLOW
TR/MULT
TR/MULT
TR/MULT
ERR/FLOW
ERR/FLOW
TR/MULT
TR/MULT
Network Device
App to App
Host to Host
Interface to Interface

Summary


Resilience
Low MTTR
Automated Response
Brittleness
Human Response
Process
High MTBF
Skills
Tools

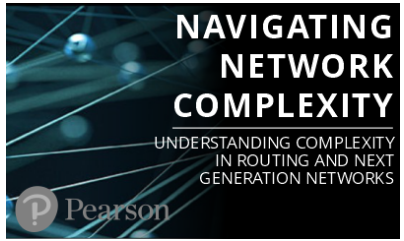
Summary

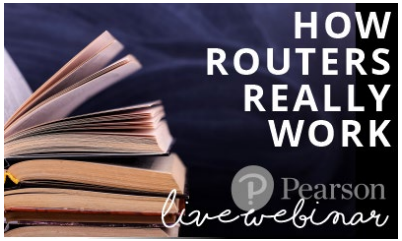
- Avoid the narrows
  - Record where you've been
  - Be willing to back up if needed
  - Stay focused on the problem you are trying to solve



observe/orient
what should I measure?
how can I measure it?
how might measuring change things?
what should normal look like?
why might it look like this?

decide/act
should I go left or right?
should I go in or out?





RULE11·TECH

Pearson