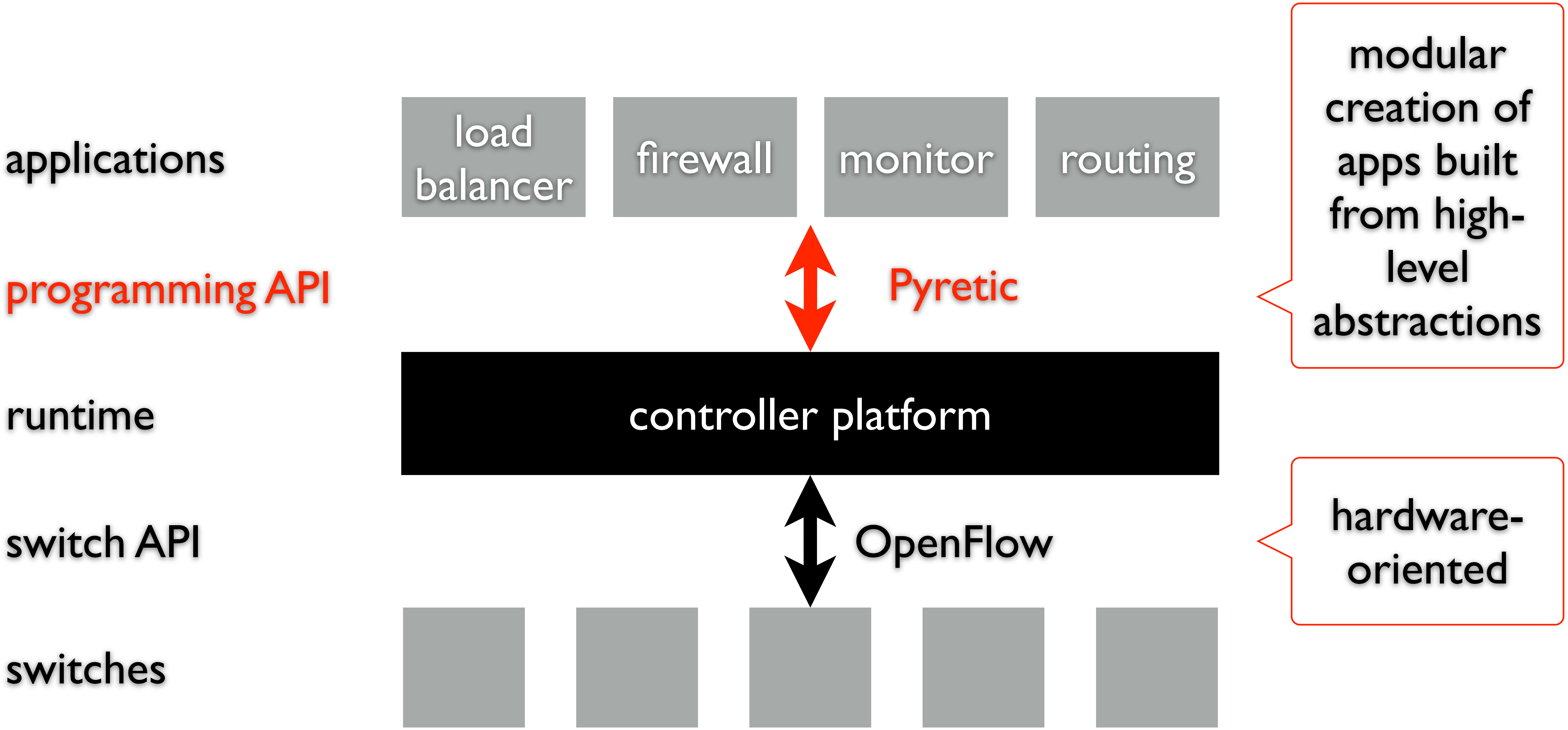
state management

5590: software defined networking

anduo wang, Temple University

T 17:30-20:00

OpenFlow, Pyretic



datacenter network (DCN)

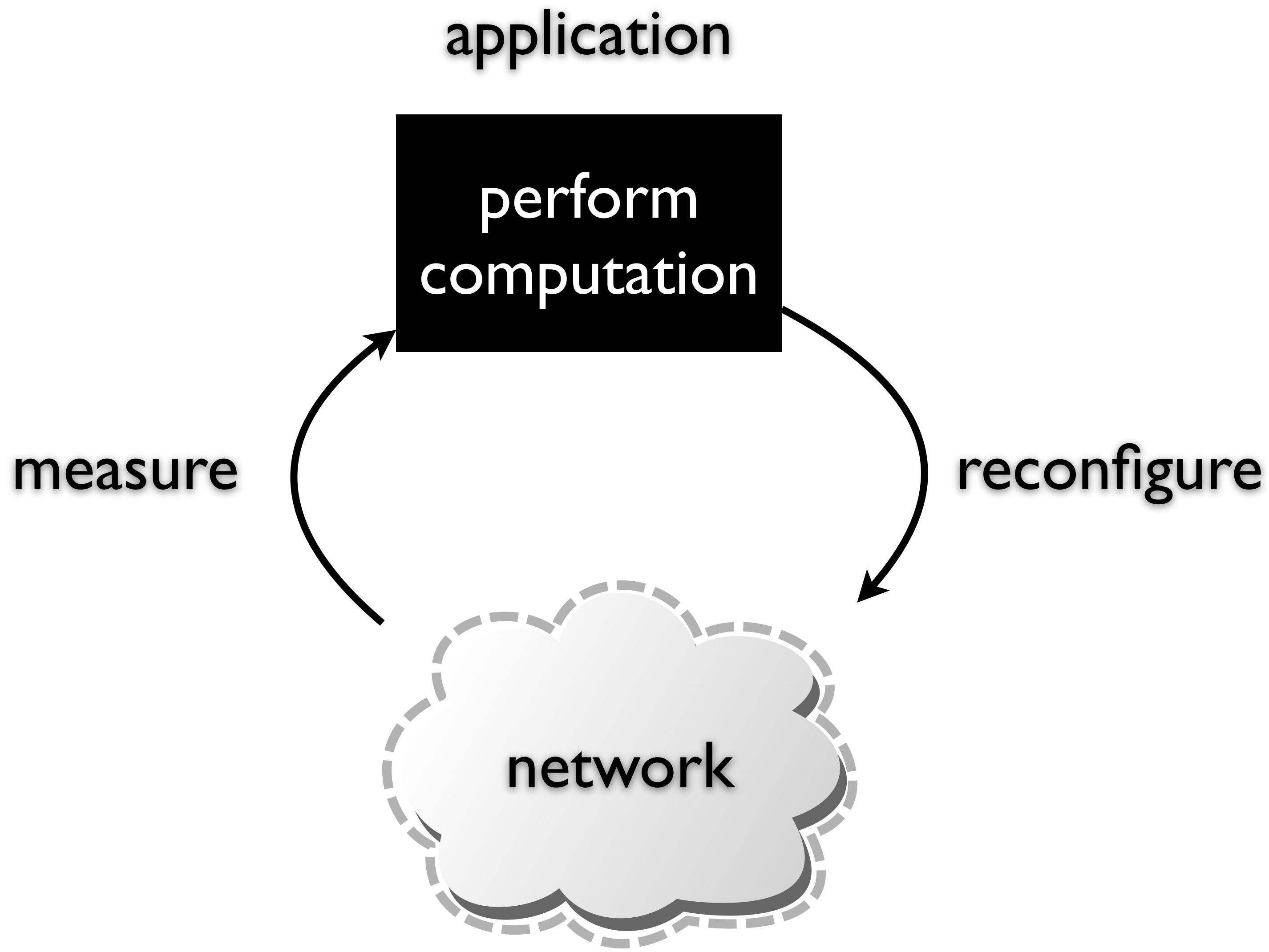
runs multiple management applications

- traffic engineering
- server load balancing
- network virtualization

infrastructure

- failure recovery *NetPilot [SIGCOMM'12]*
- energy saving *Elastic tree [NSDI'10]*
- switch configuration

management applications



application

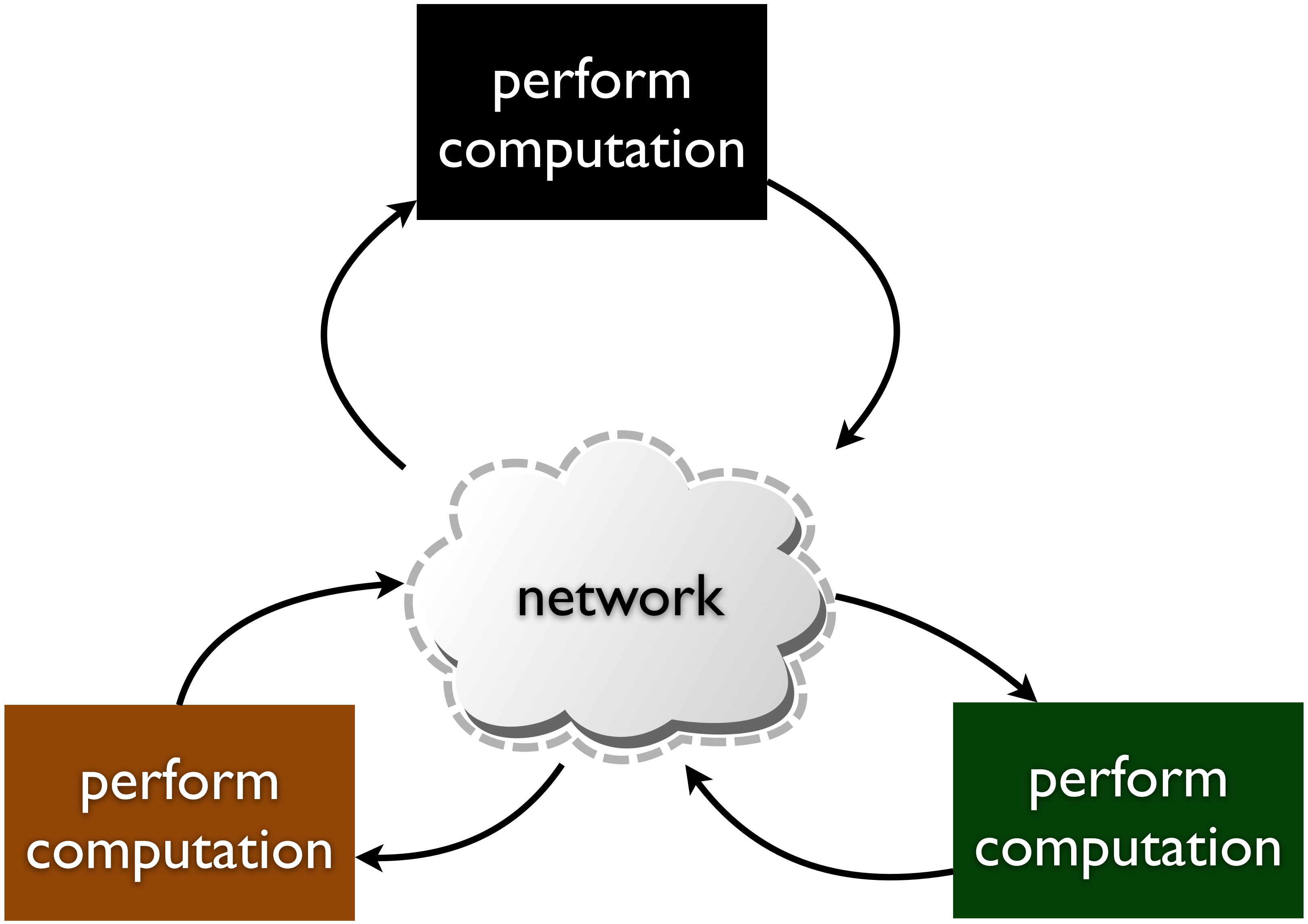
perform computation

measure

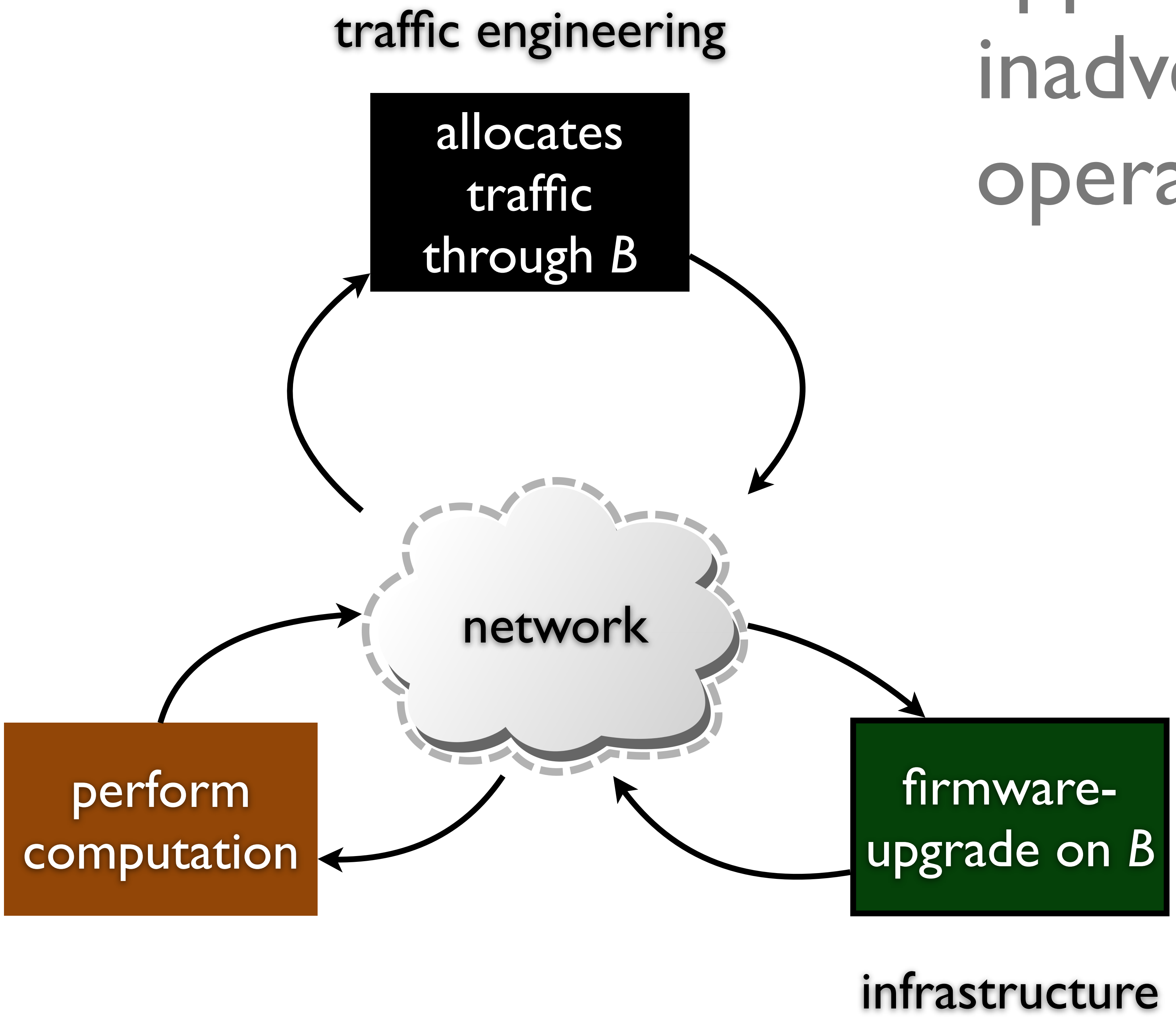
reconfigure

network

running multiple applications

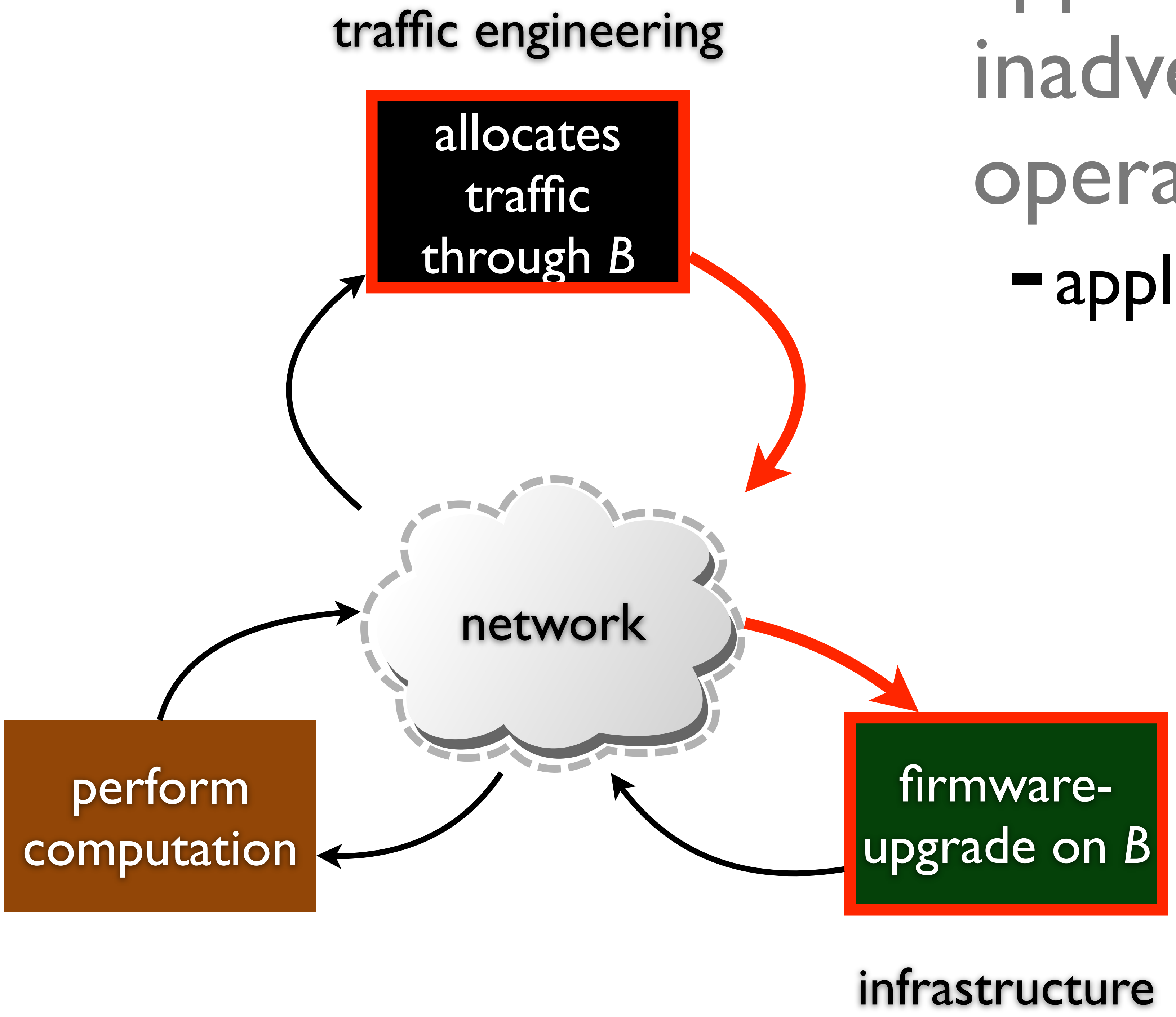


running multiple applications



applications can inadvertently affect the operations of another

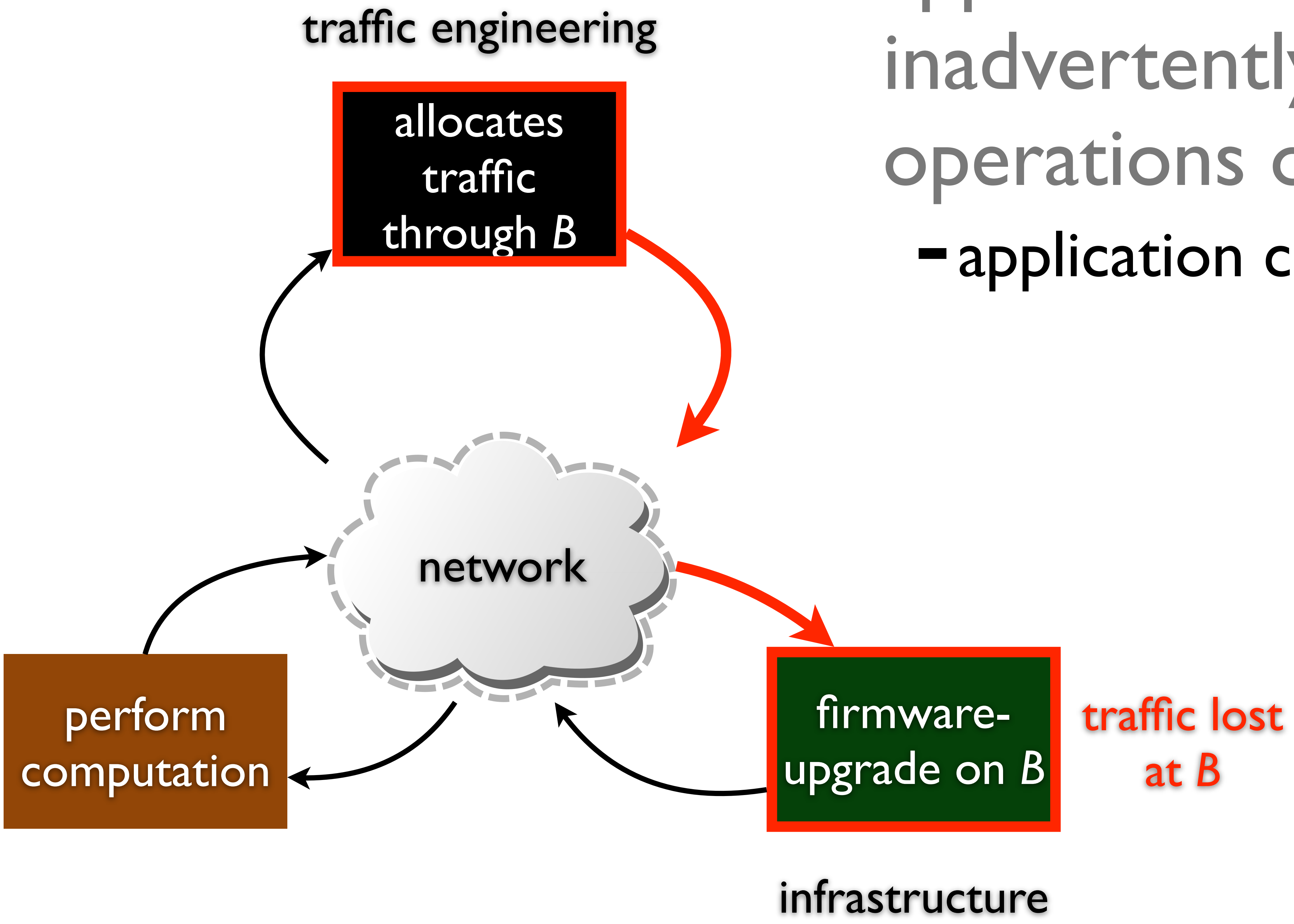
running multiple applications



applications can inadvertently affect the operations of another

- application conflict

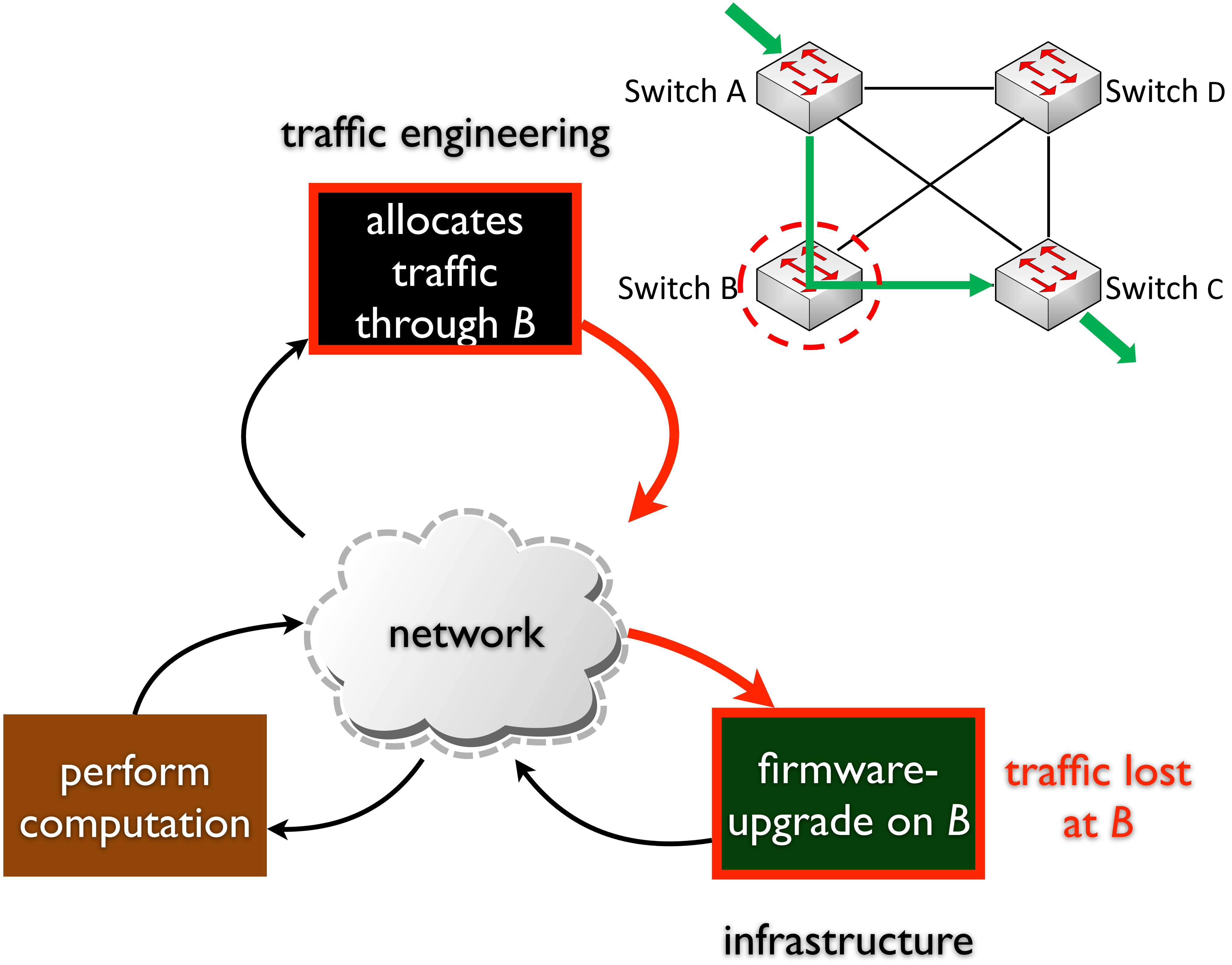
running multiple applications



applications can inadvertently affect the operations of another

- application conflict

running multiple applications



TE application allocates traffic on the path through Switch B

Switch-firmware-upgrad
application schedules switch B for upgrading

traffic engineering

allocates traffic through *B*

network

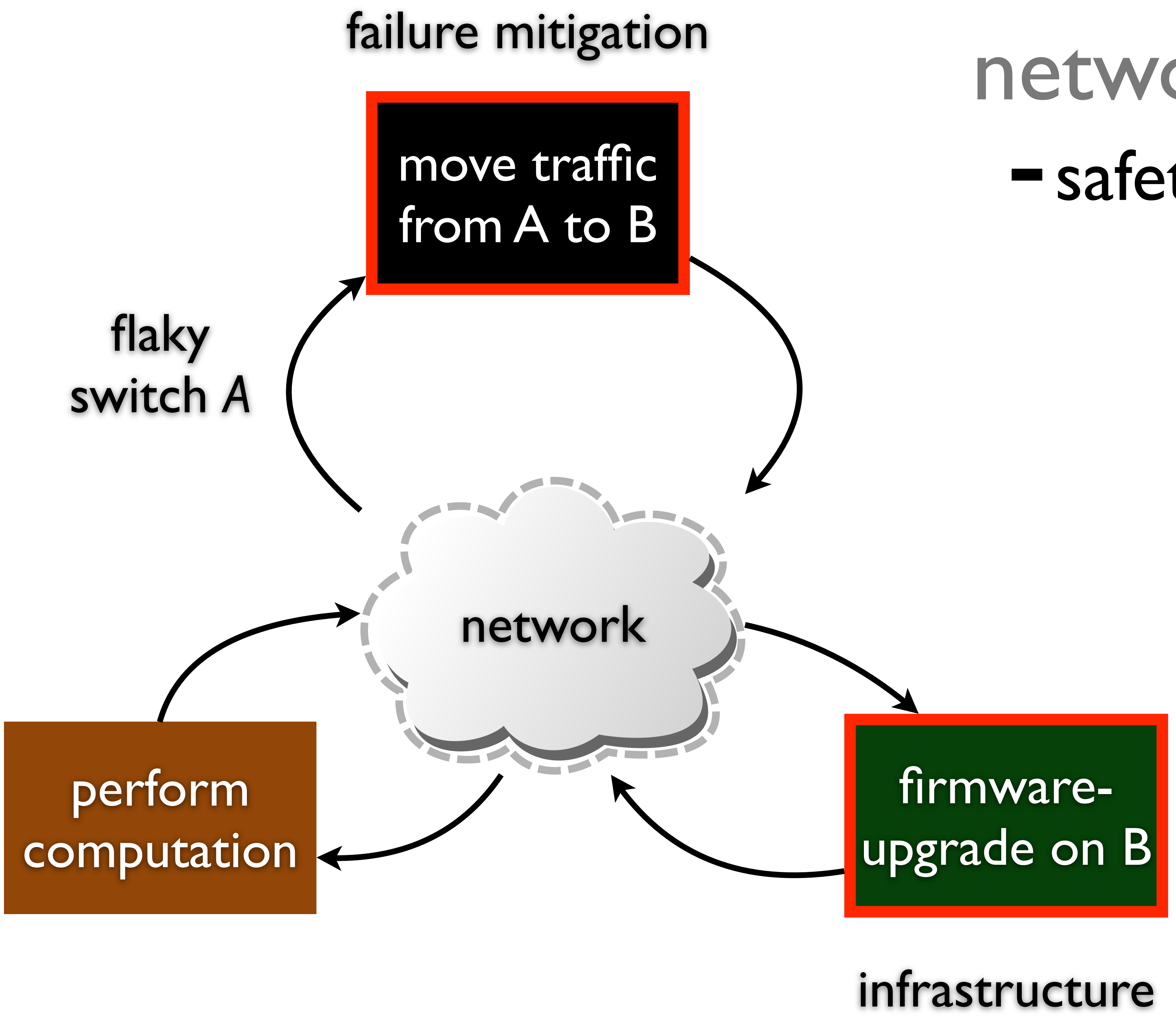
perform computation

firmware-upgrade on *B*

traffic lost at *B*

infrastructure

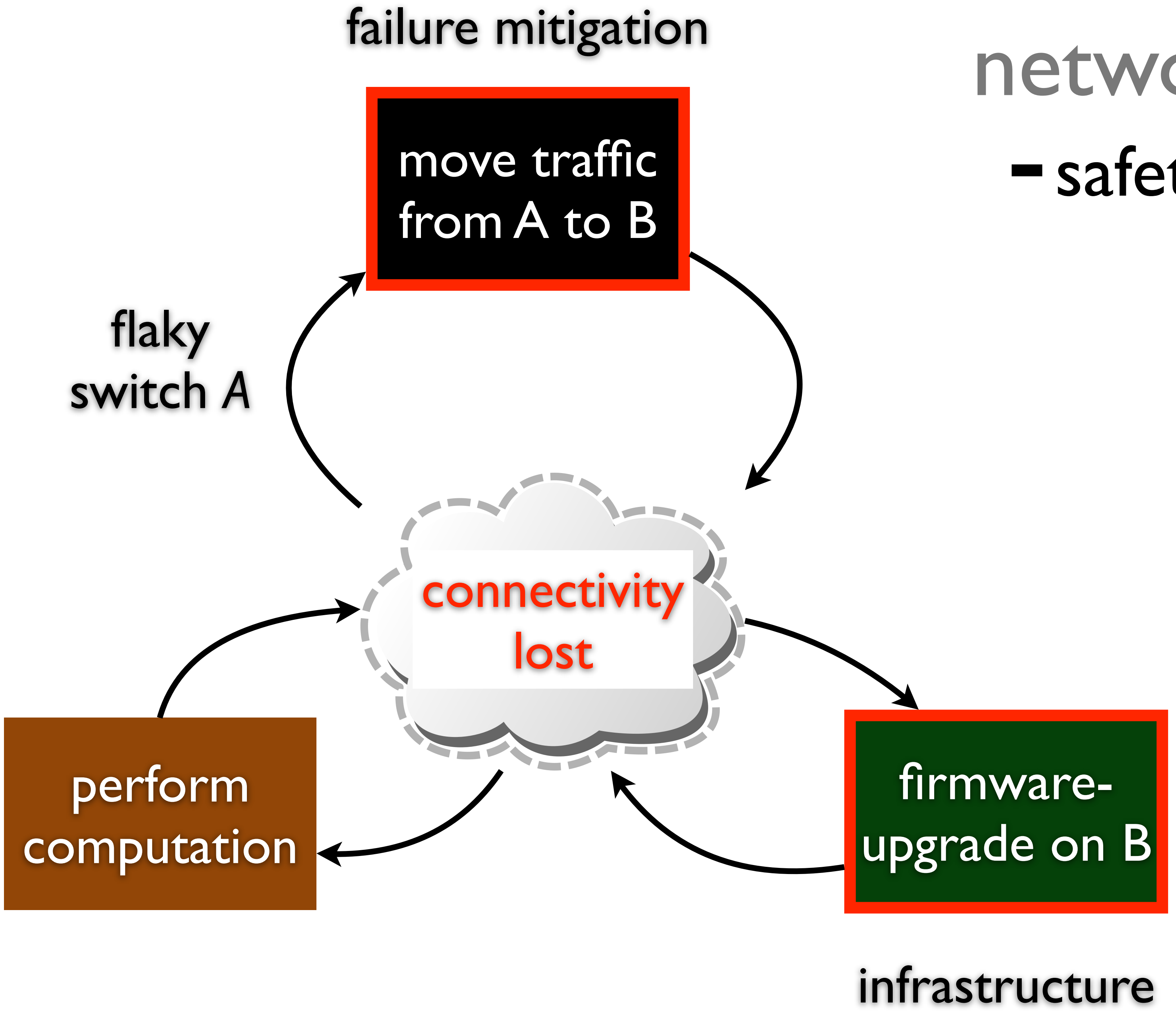
running multiple applications



combined effects lead to network-wide failures

- safety failure

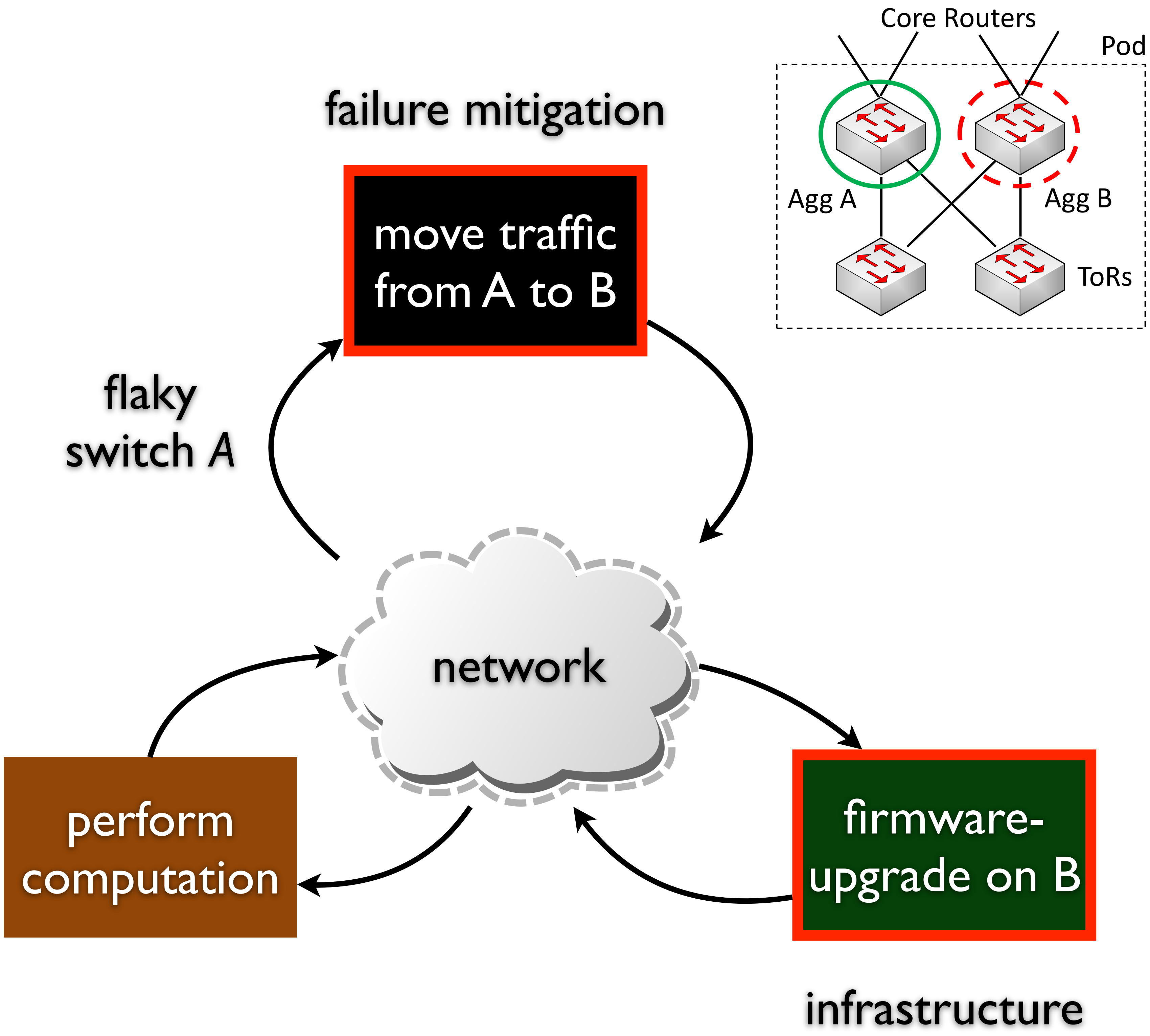
running multiple applications



combined effects lead to network-wide failures

- safety failure

running multiple applications



failure mitigation

move traffic from A to B

flaky switch *A*

network

perform computation

firmware-upgrade on B

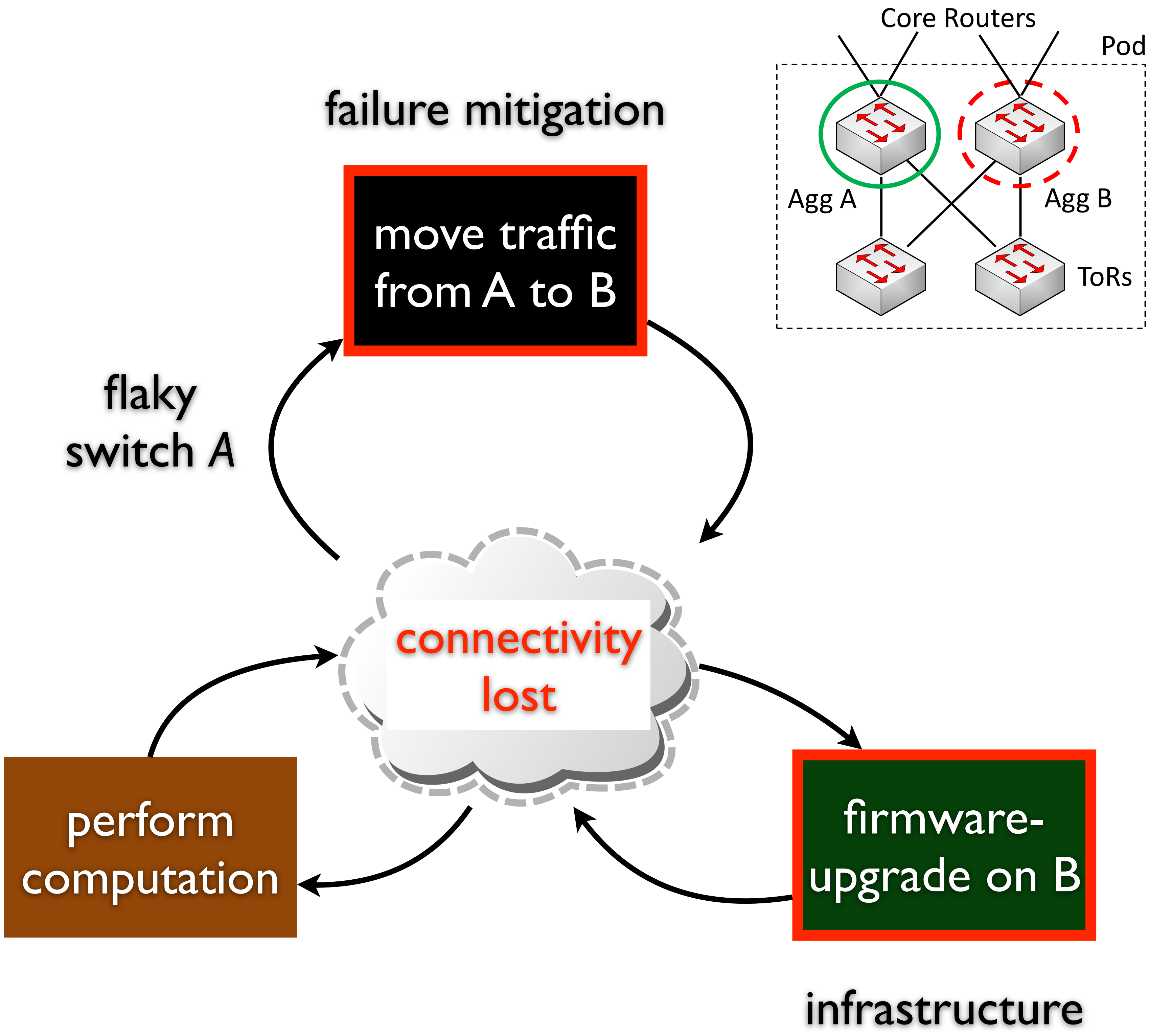
infrastructure

Failure-mitigation application tries to shut down flaky Agg A, assuming Agg B is up.

Switch-firmware-upgrade application assumes Agg A is up, and schedules Agg B for upgrading.

Connectivity of ToRs is lost

running multiple applications



failure mitigation

move traffic from A to B

flaky switch *A*

connectivity lost

perform computation

firmware-upgrade on B

infrastructure

Failure-mitigation application tries to shut down flaky Agg A, assuming Agg B is up.

Switch-firmware-upgrade application assumes Agg A is up, and schedules Agg B for upgrading.

Connectivity of ToRs is lost

alternative to running multiple applications

one single monolithic application

- complex
- explicit coordination
- high overhead on applications

alternative to running multiple applications

one single monolithic application

- complex
- explicit coordination
- high overhead on applications

tightly coupled, repeated extension

statesman approach

statesman approach

build and run applications in a loosely coupled manner

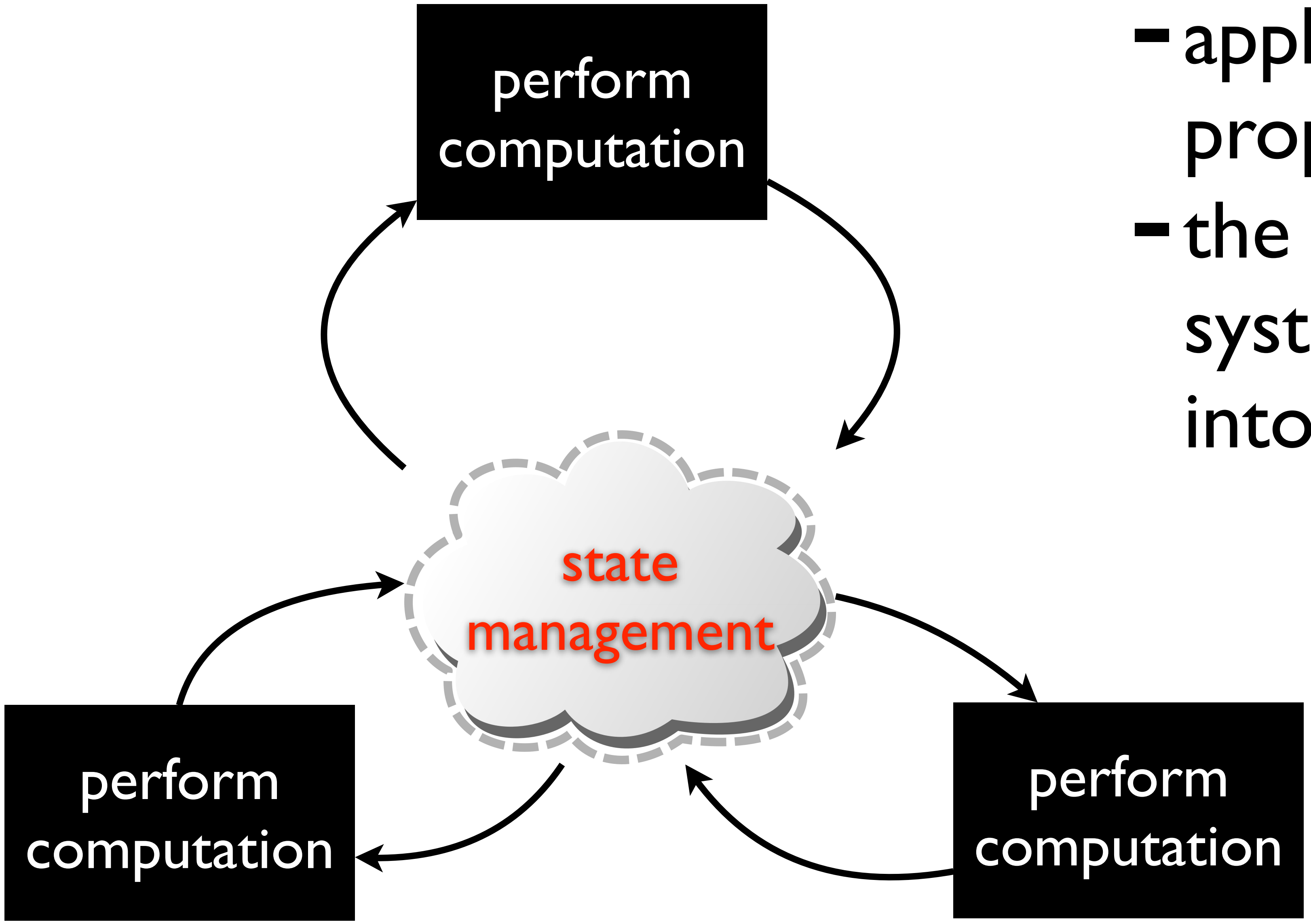
statesman approach

build and run applications in a loosely coupled manner

introduce a separate (state) management system

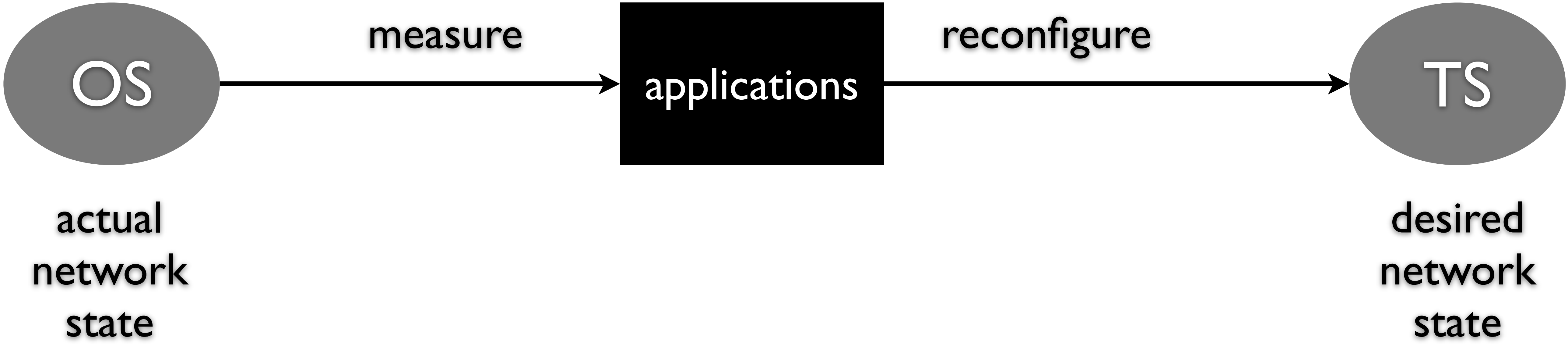
- conflict resolution
- invariant enforcement

statesman approach

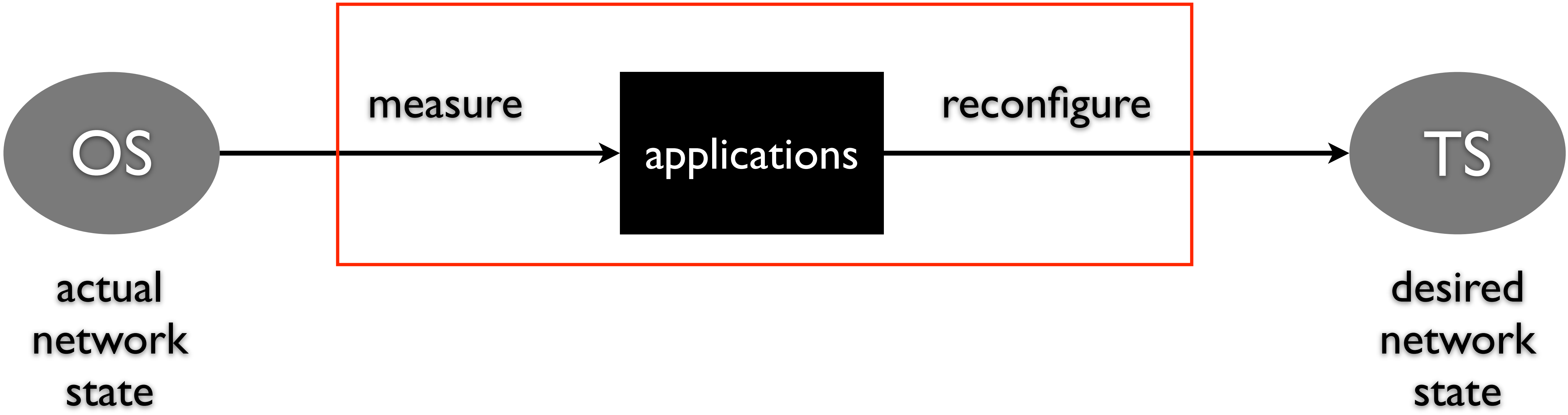


- applications "pull" observed states (OS)
- applications "push" proposed states (PS)
- the separate statesman system "merges" the states into target states (TS)

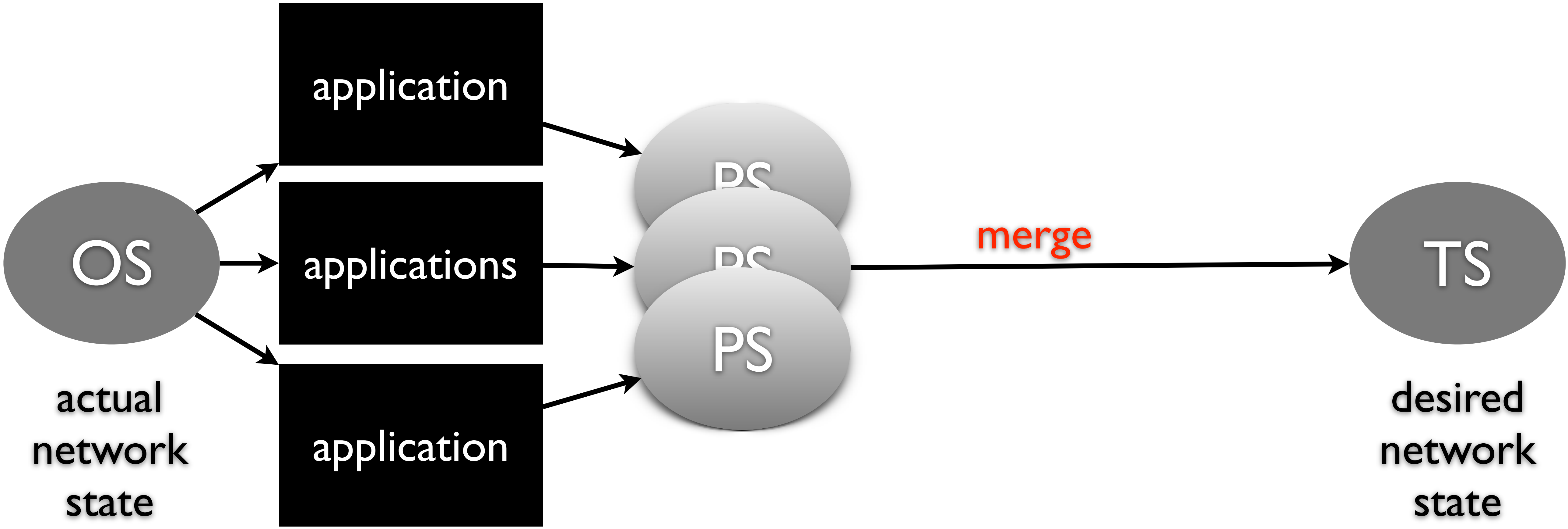
statesman approach



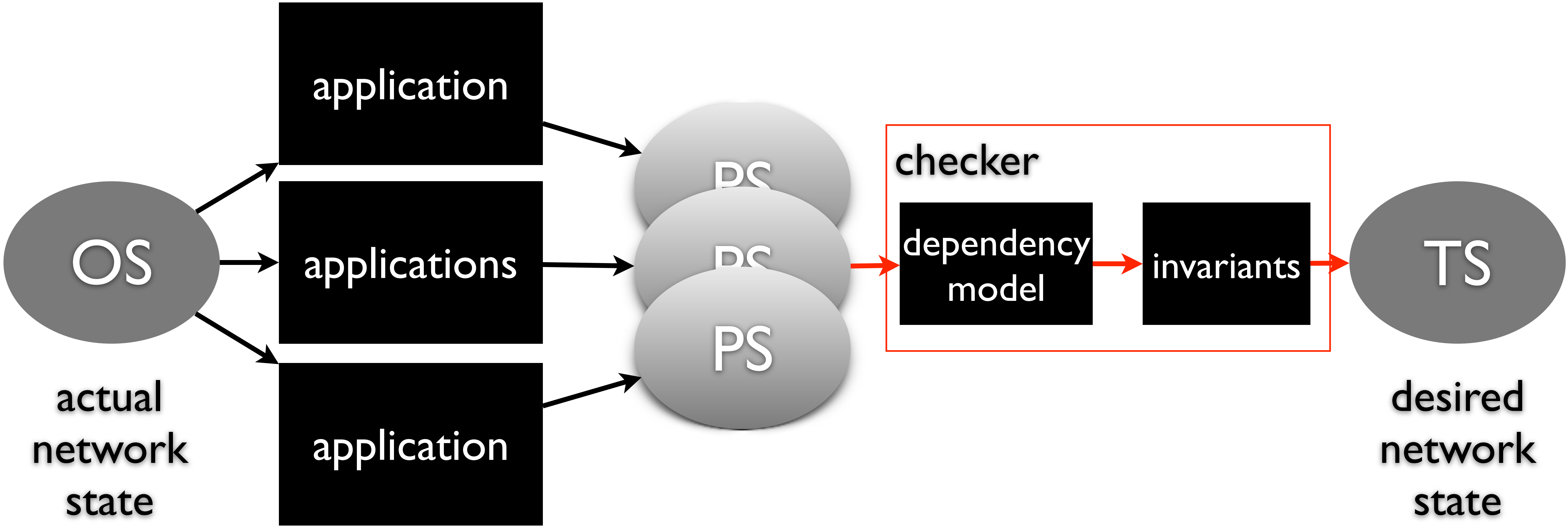
statesman approach



statesman approach



statesman approach



checker

checker

use state dependency model

- determine whether PSes applicable to existing OSes
- detect and resolve conflicts among PSes
- form TSes

checker

use state dependency model

- determine whether PSes applicable to existing OSes
- detect and resolve conflicts among PSes
- form TSes

use operator-specified invariants

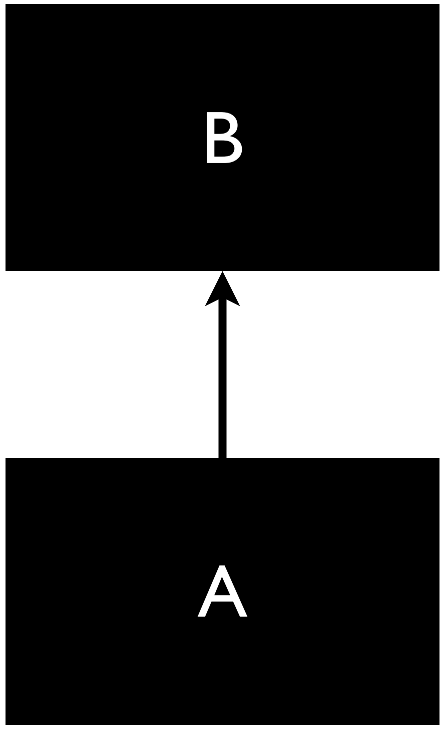
- examine the TSes against the invariants

state dependency model

state variables in one application's PS can depend on state variables in another application's PS

B depends on A

- A is a prerequisite for writing B states

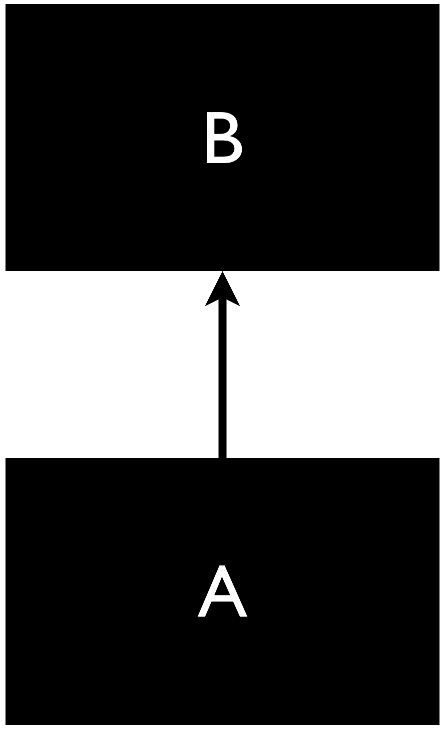


state dependency model

state variables in one application's PS can depend on state variables in another application's PS

B depends on A

- A is a prerequisite for writing B states
- B is controllable only if A value is appropriate

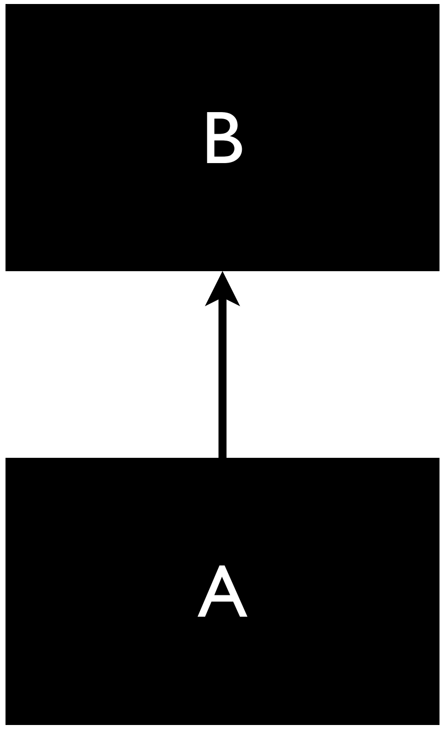


state dependency model

state variables in one application's PS can depend on state variables in another application's PS

B depends on A

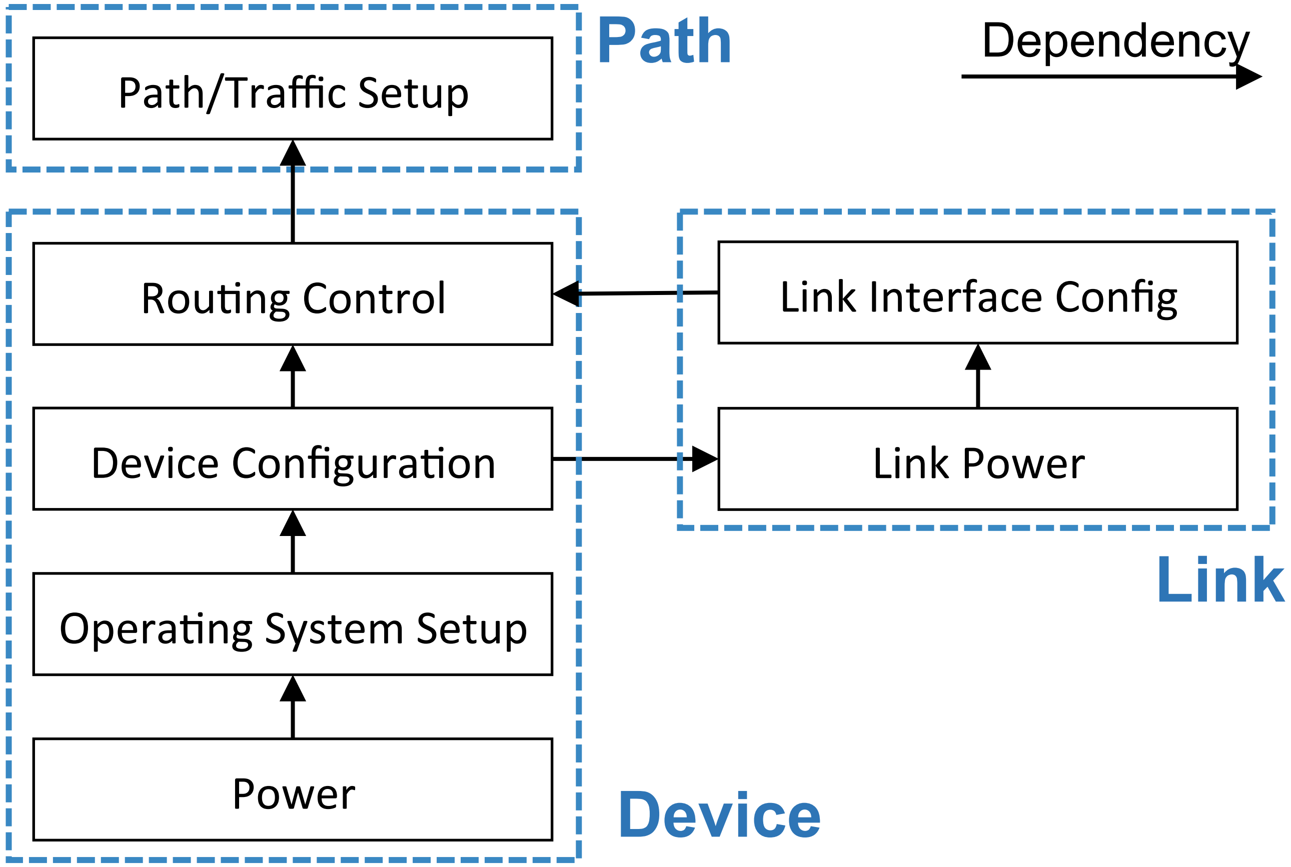
- A is a prerequisite for writing B states
- B is controllable only if A value is appropriate



conflicts

- B is uncontrollable due to state (or state change) in A

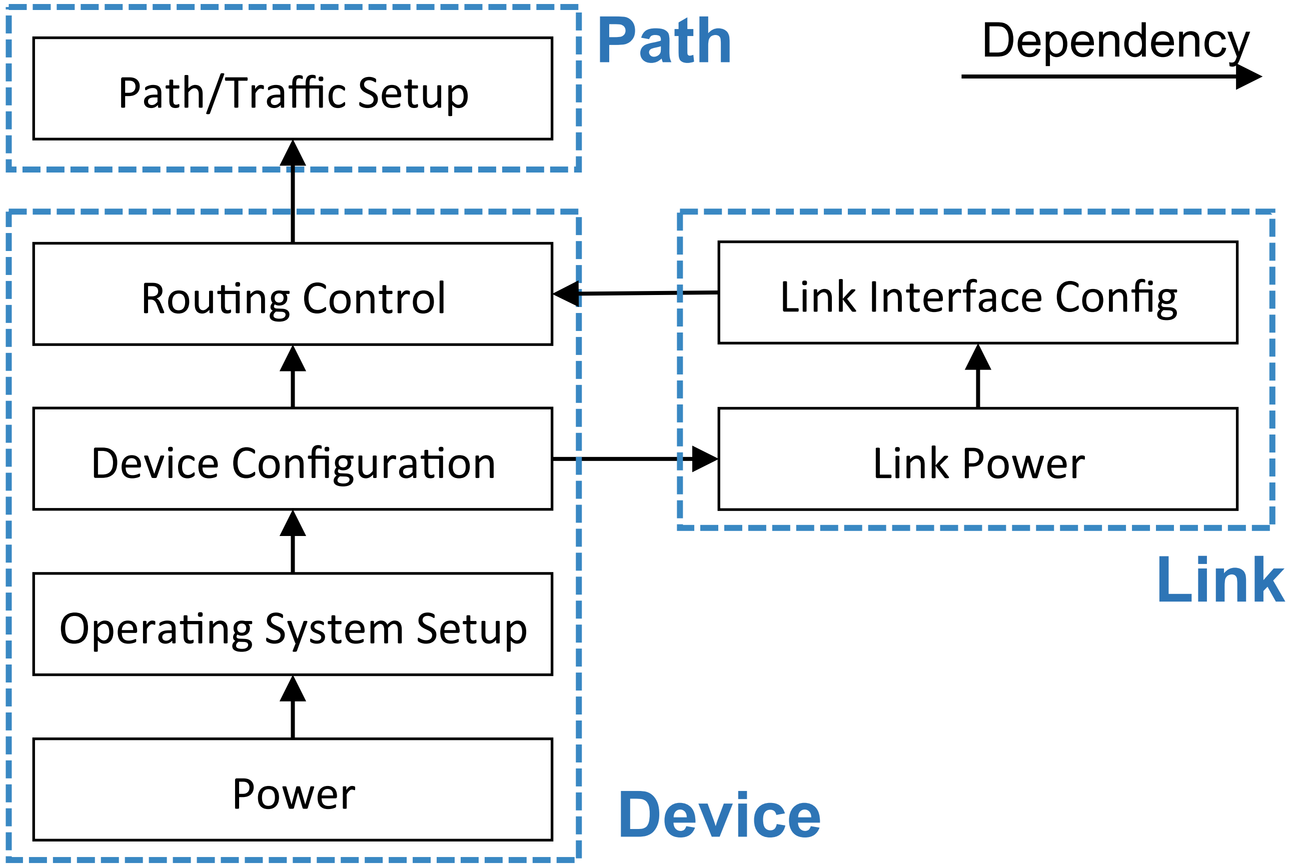
using state dependency model



using state dependency model

Statesman exposes

- B's (latest) value
- together with a logical controllability variable
  - "1" only if all B's parents are controllable



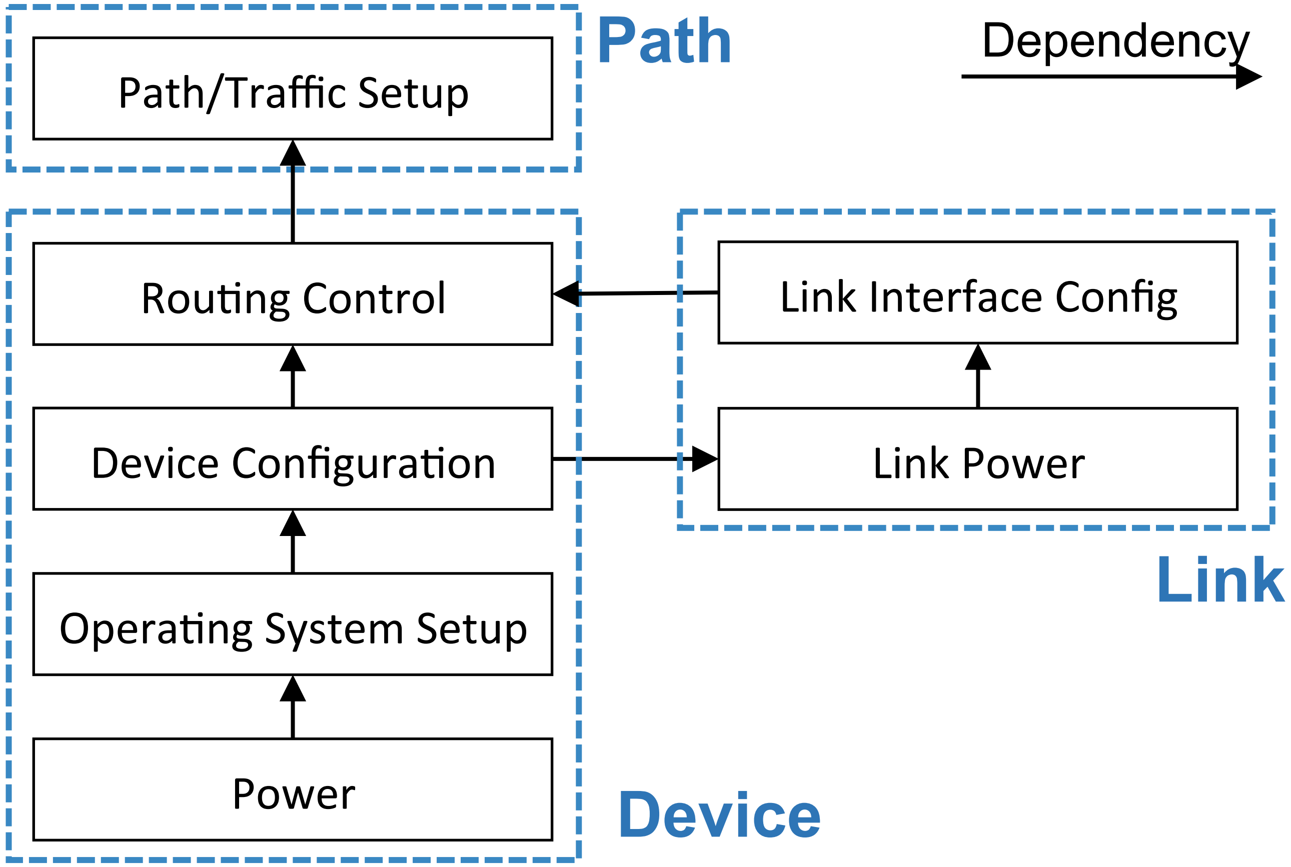
using state dependency model

Statesman exposes

- B's (latest) value
- together with a logical controllability variable
  - "1" only if all B's parents are controllable

question

- how to extend the dependency model?
- advantage of having an explicit separate model?



resolving conflicts

resolving conflicts

TS-OS, PS-OS

- conflicts due to the changing OS
  - makes some variables in TS/PS uncontrollable
- solution: simply reject

resolving conflicts

TS-OS, PS-OS

- conflicts due to the changing OS
  - makes some variables in TS/PS uncontrollable
- solution: simply reject

PS-TS

- a PS can conflict with the TS due to an accepted PS from another application
  - TS is really just the accumulation of all accepted in the past
- solution
  - accept with last-write-wins / priority-based locking
  - at the level of individual switches and links

maintaining invariants

what

- invariants: infrastructure's operational stability, independent of apps
  - suffice to safeguard the network & not too stringent with app goals
- examples: connectivity, capacity

how

- checking TS against invariants
  - maintain a base network state graph using values from the OS
  - compute difference between TS and OS
  - check invariants on the new network state

discussion

making multiple applications coexist

- ONIX, NOX: no support
- Pyretic, Pane, Maple: compose target traffic management applications
- Corybantics: hosting multiple applications on isolated slices

statesman: use cases, evaluations …

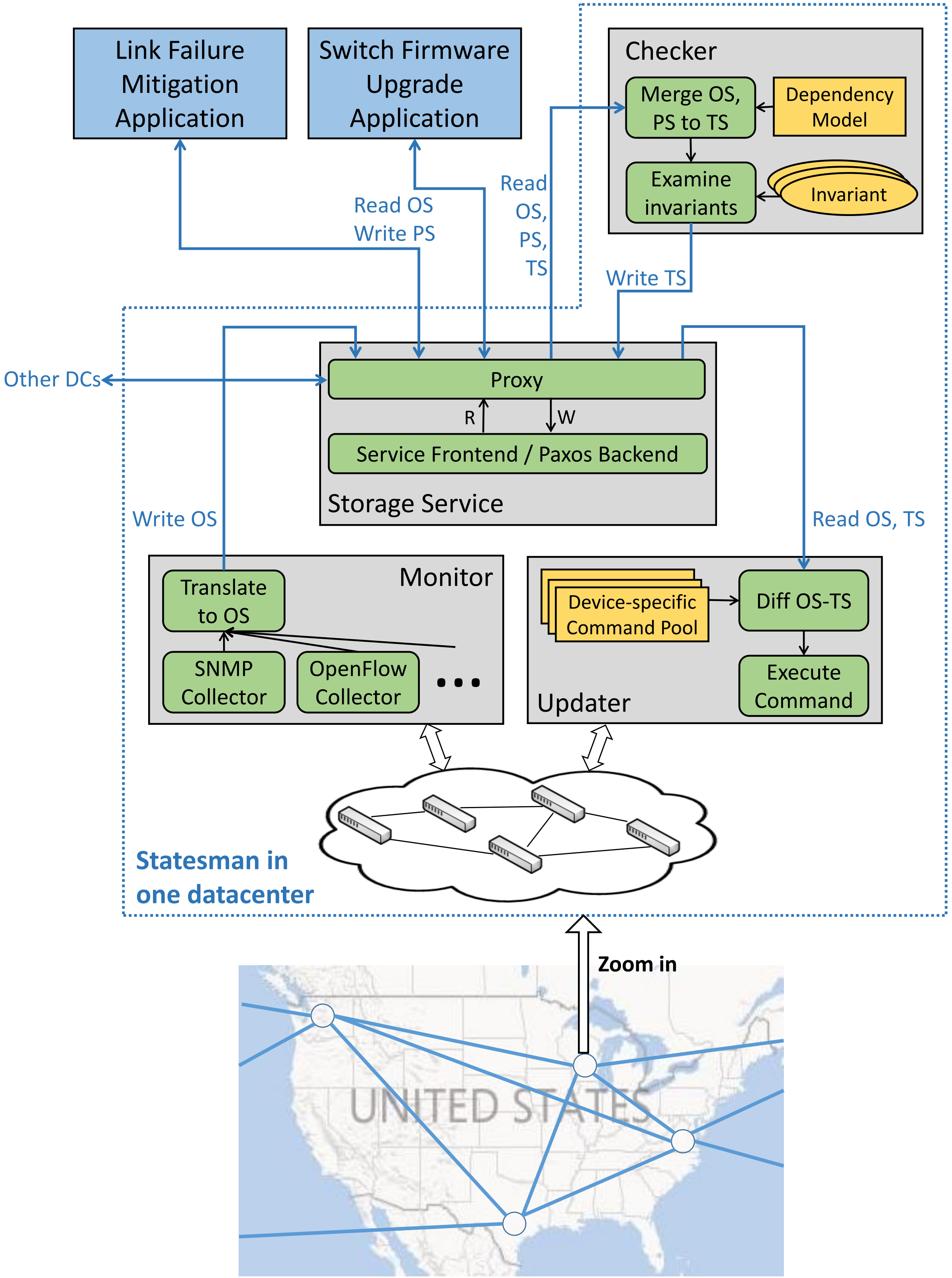
statesman deployment

10 geographically-distributed datacenters (DCs)

- cover switches, links within each DC and across DC (WAN)

three applications

- switch-upgrade
- failure-mitigation
- inter-DC TE



challenges—maintaining globally available and distributed states

- inter-DC
  - due to WAN failures, DCs may be disconnected
- within-DC
  - huge volume of state data: hundreds of thousands of switches and links
  - millions of state variables

challenges—updating DCN states

- heterogeneity: diverse range of network elements expose heterogenous interfaces for updates
- device can fail during an update
- device respond slow, dominating the application control loop

solution—maintaining globally available and distributed states

solution—maintaining globally available and distributed states

partitioning checker's responsibility into impact groups

- one impact group per DC
- one additional impact group with border routers of all DCs and the WAN links

solution—maintaining globally available and distributed states

partitioning checker’s responsibility into impact groups

- one impact group per DC
- one additional impact group with border routers of all DCs and the WAN links

partitioning monitor

- split monitor’s responsibility into many instances
  - each covers 1k switches

solution—updating DCN states

solution—updating DCN states

heterogeneity

- OpenFlow and command templates

solution—updating DCN states

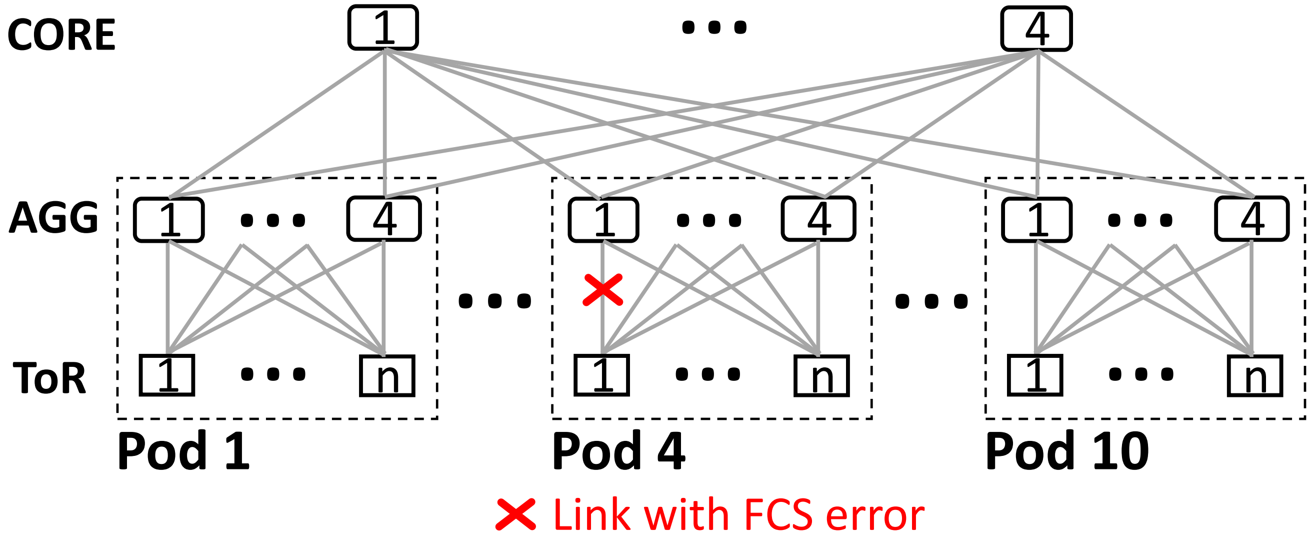
heterogeneity

- OpenFlow and command templates

dynamic failures

- stateless updates
- simply push to the devices the latest OS-TS difference

use case: maintaining invariants

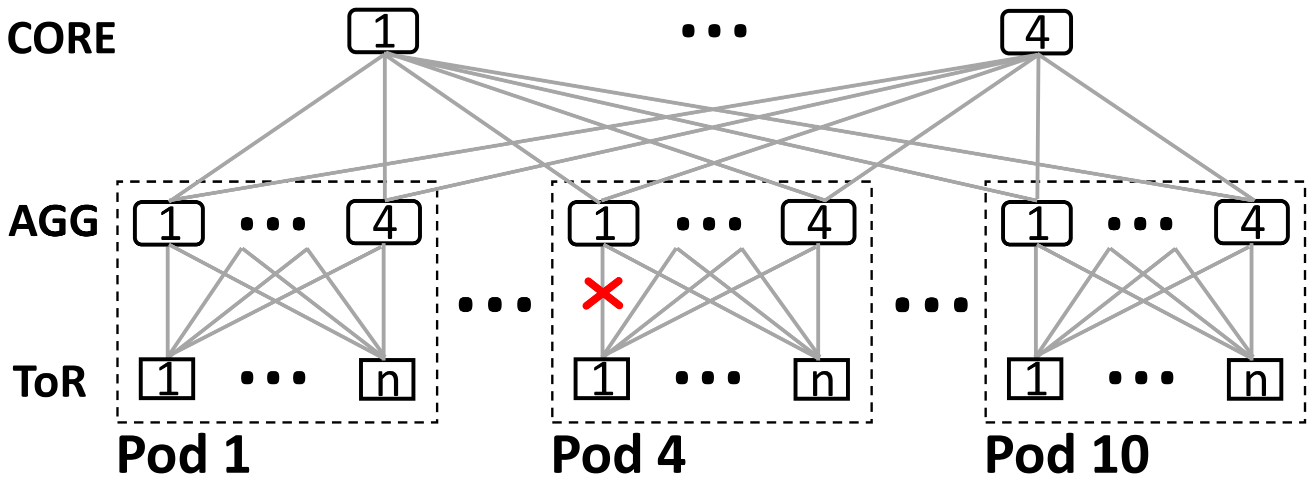


switch_upgrade and failure_mitigation coexist

statesman goal: maintaining capacity invariant

- 99% ToR pairs have at least 50% capacity

use case: maintaining invariants



✕ Link with FCS error

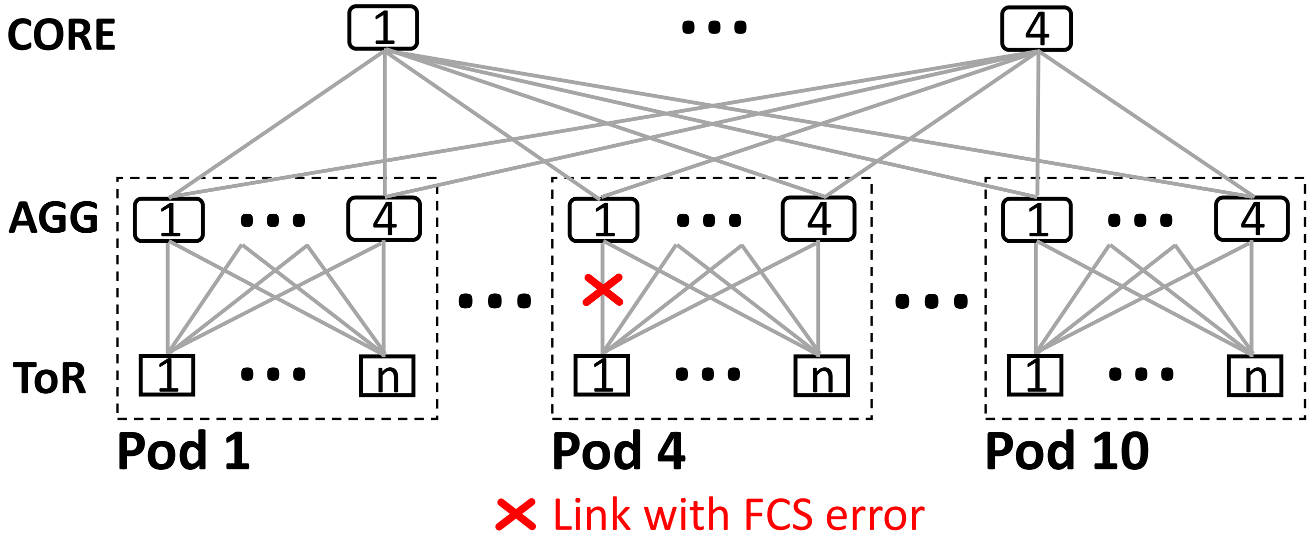
one DC with 10 pods

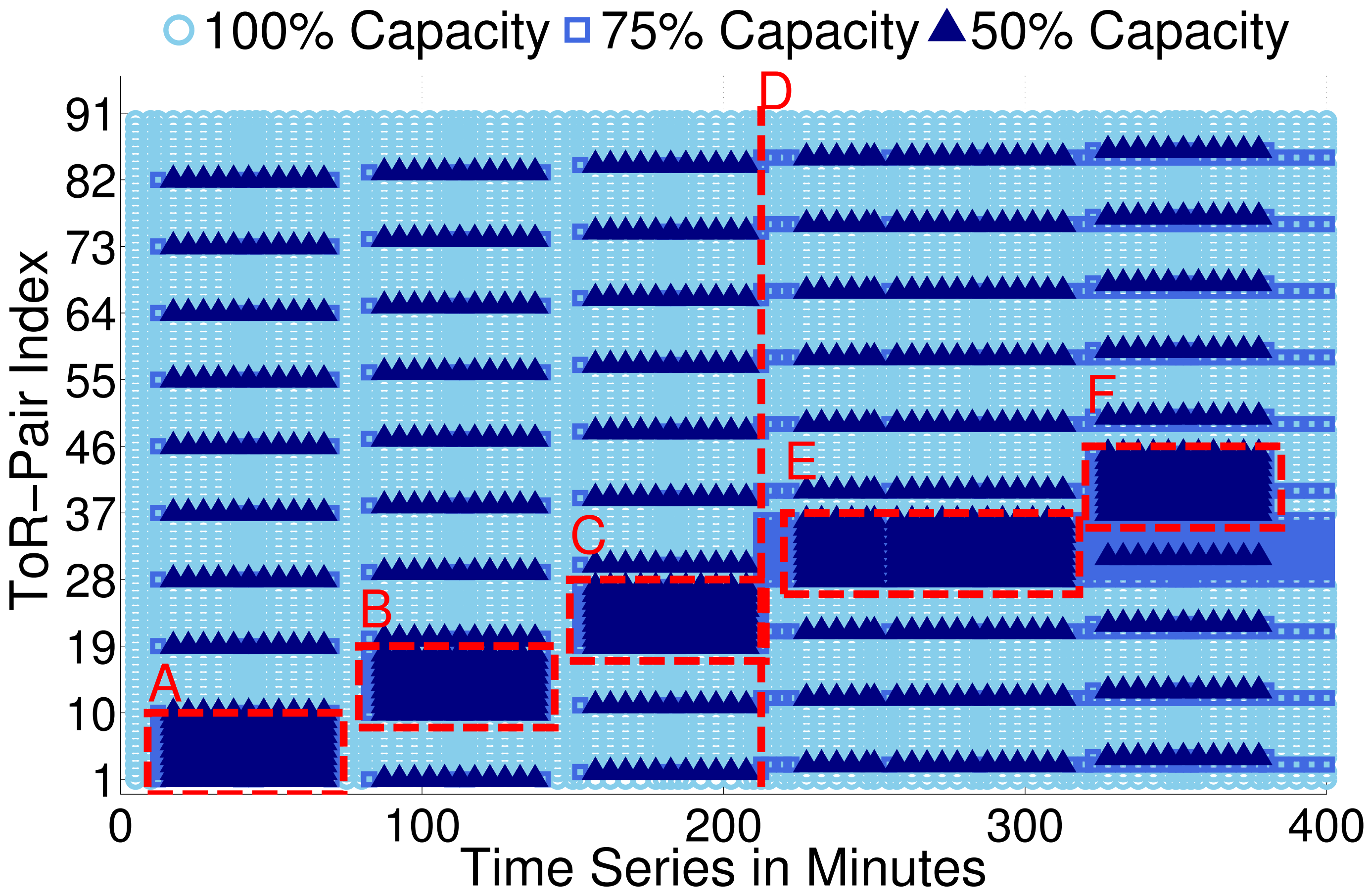
- each pod has 4 AGGs and a number of ToRs

switch_upgrade

- upgrade all 40 AGGs
- (sequentially) pod by pod
- attempt parallel upgrades within each pod

use case: maintaining invariants

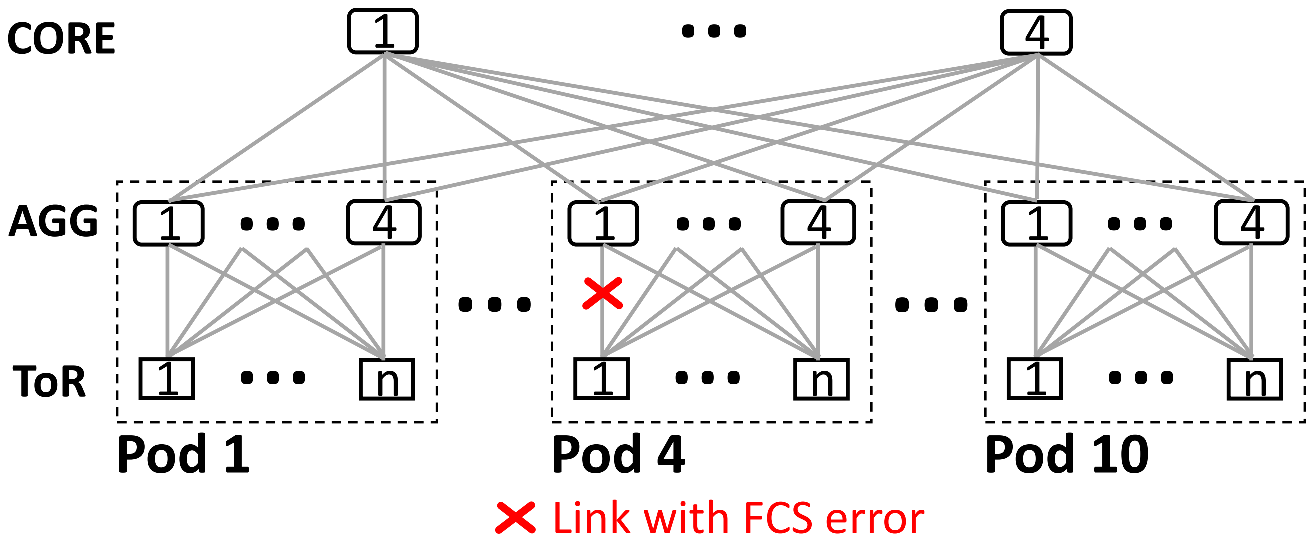


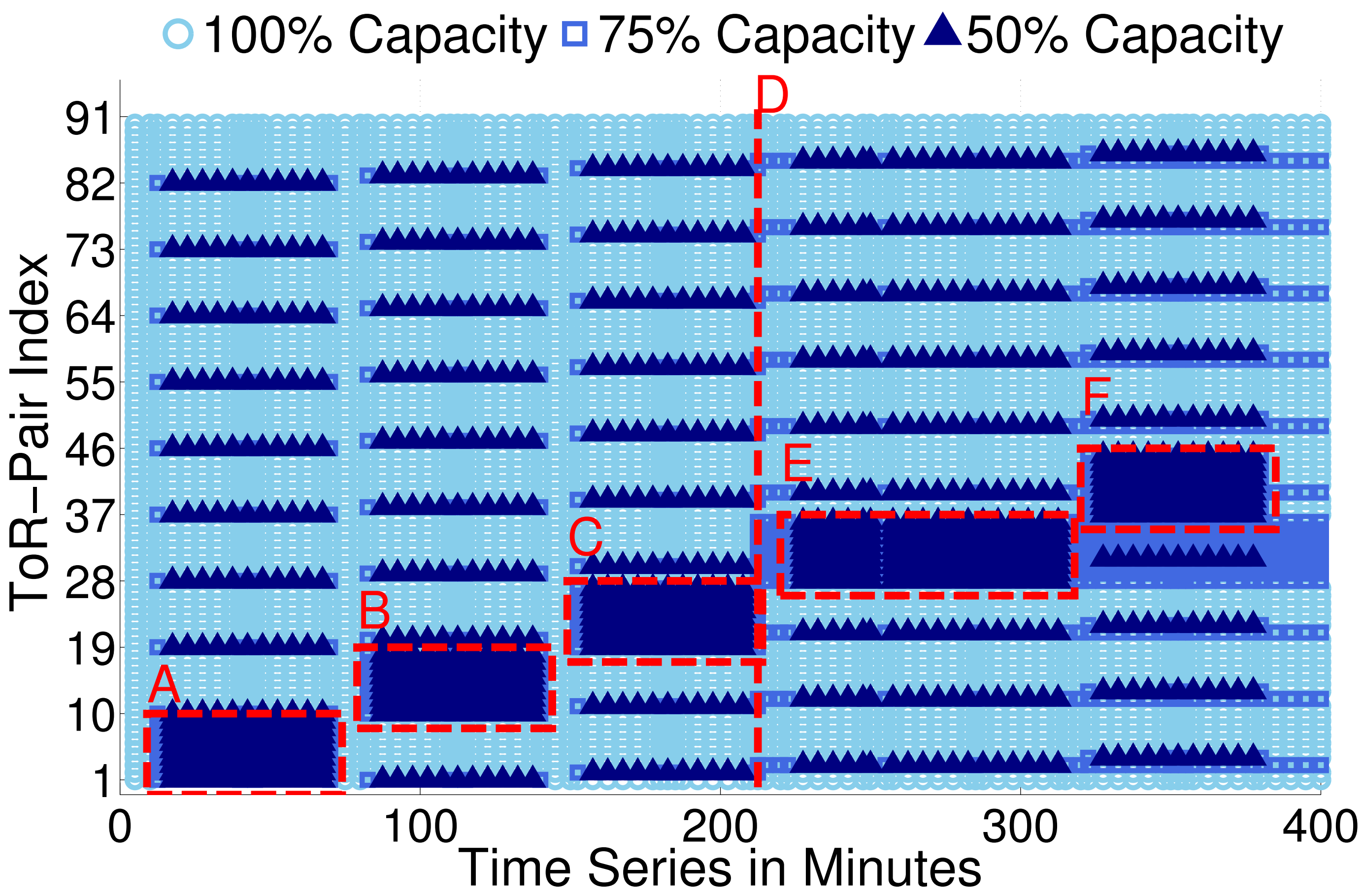


90 ToR pairs

- one ToR from each pod
- put the 9 ToR pairs from the same pods together

use case: maintaining invariants





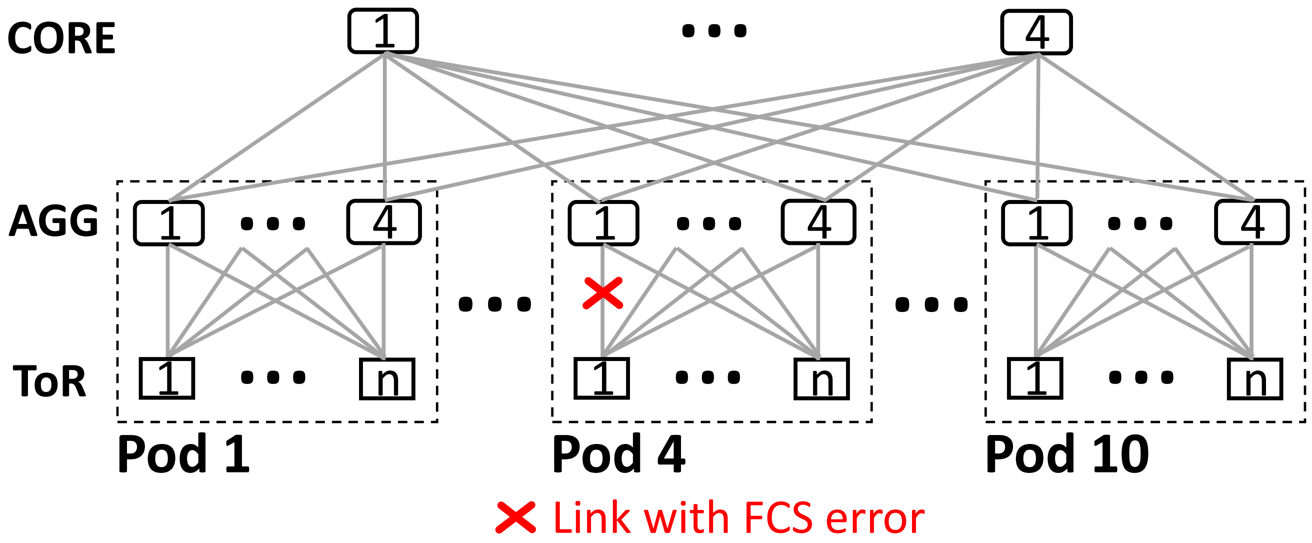
90 ToR pairs

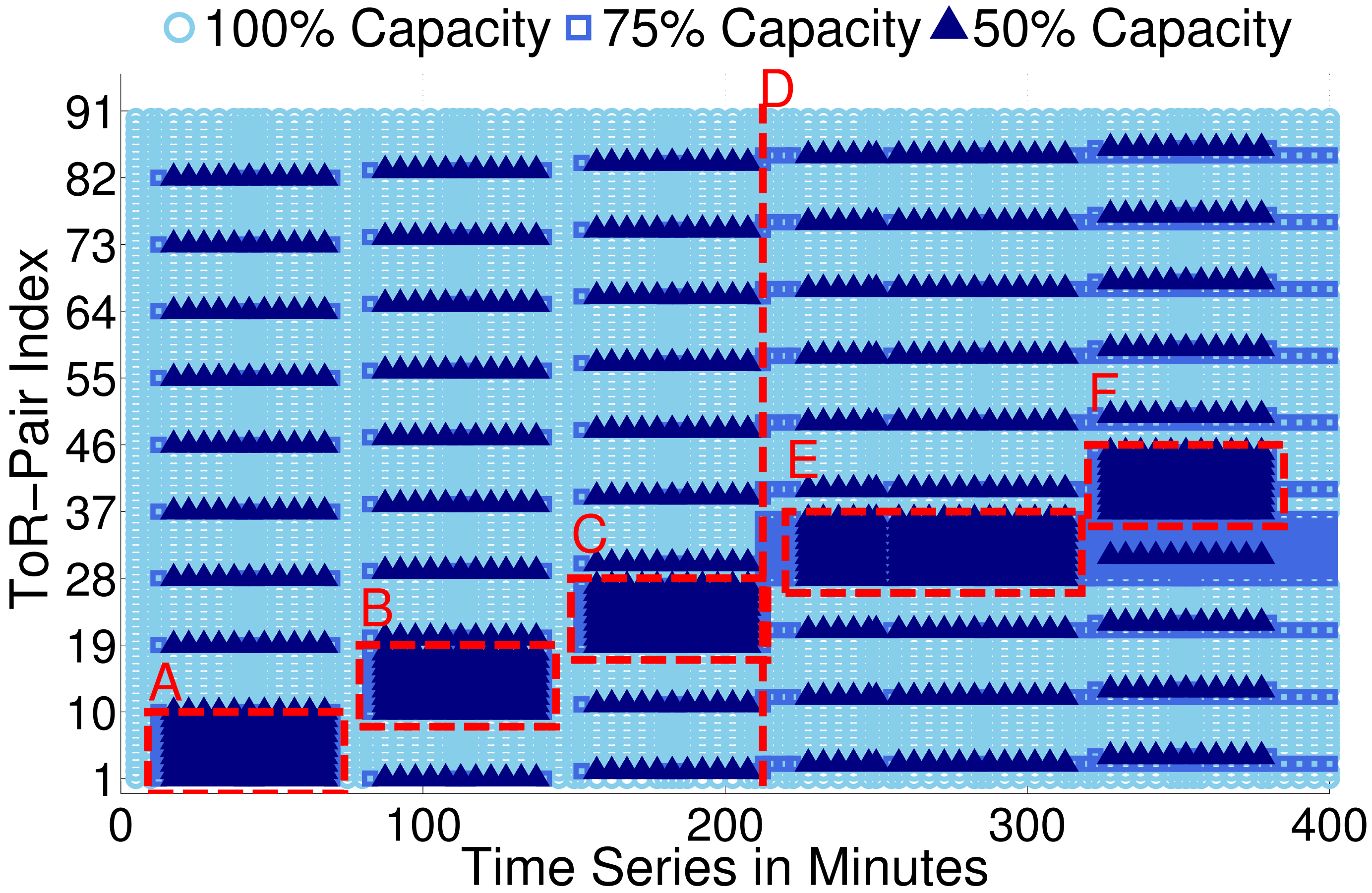
- one ToR from each pod
- put the 9 ToR pairs from the same pods together

A,B,C

- switch-upgrade upgrades pod 1,2,3

use case: maintaining invariants





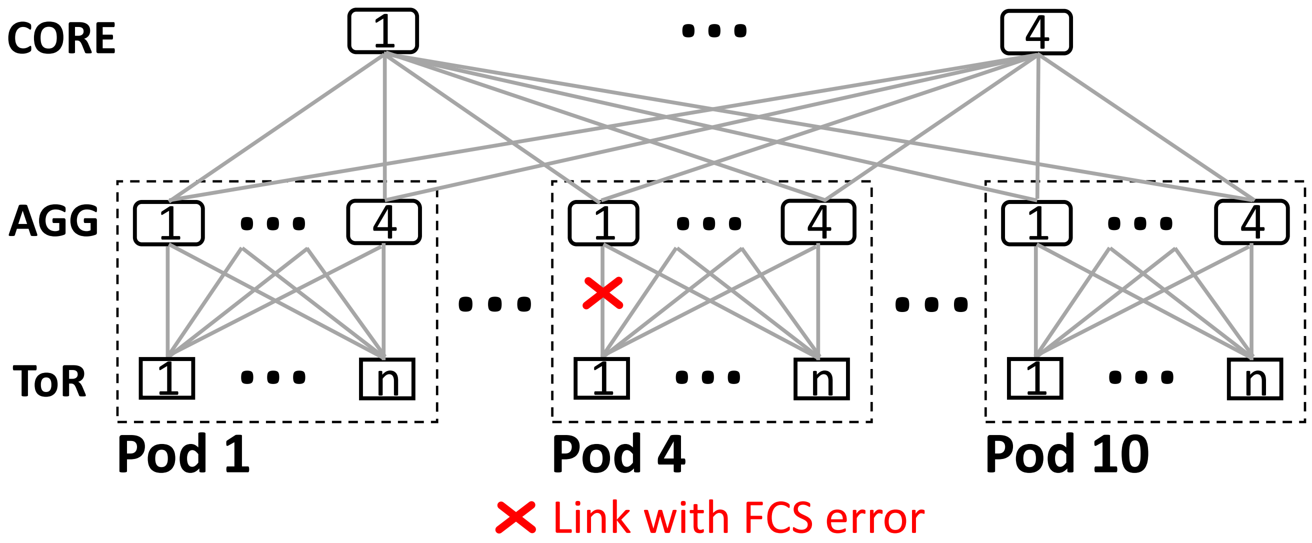
90 ToR pairs

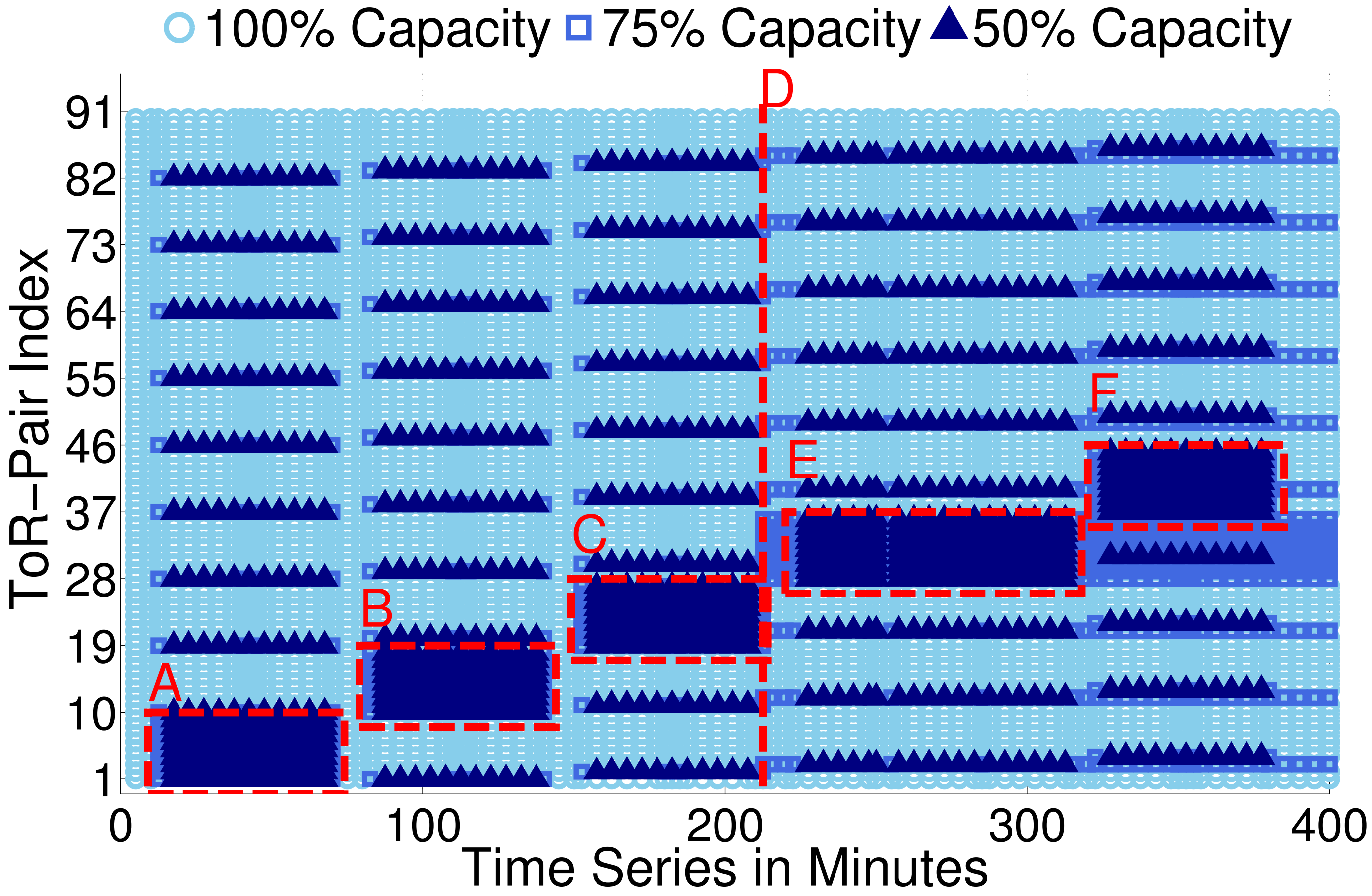
- one ToR from each pod
- put the 9 ToR pairs from the same pods together

D

- failure-mitigation detects problem, shutting down link

use case: maintaining invariants





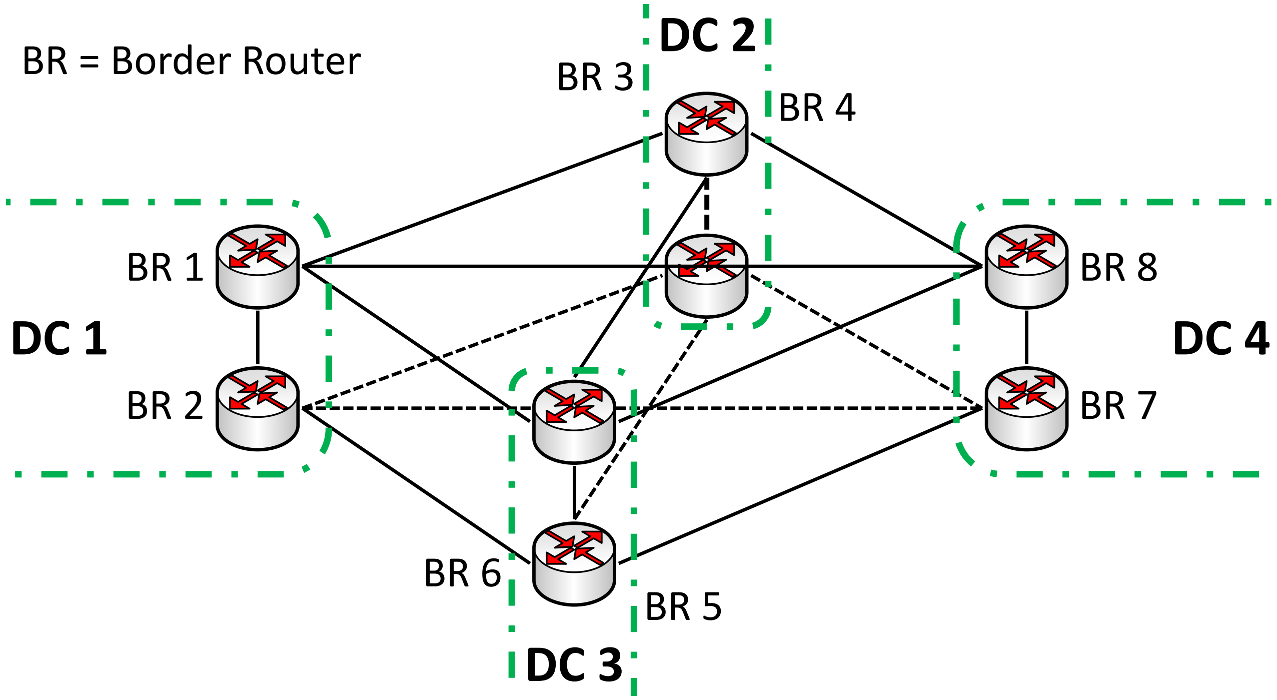
90 ToR pairs

- one ToR from each pod
- put the 9 ToR pairs from the same pods together

E

- switch-upgrade slows down by the checker
  - upgrades Agg 1,2 together; then Agg 3; finally Agg 4

use case: resolving conflicts



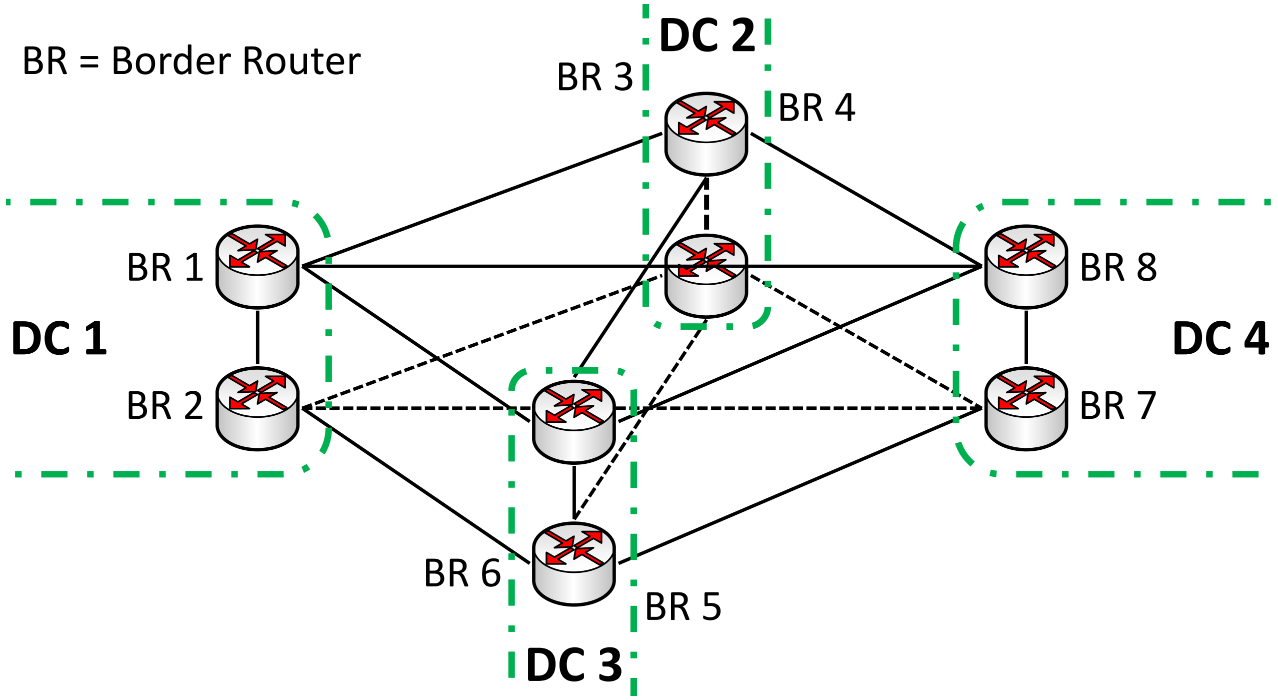
setup

- 8 border routers (BRs)
- 24 (12 physical links x 2 directions) inter-DC links

goal

- upgrade BRs while inter-DC TE is on

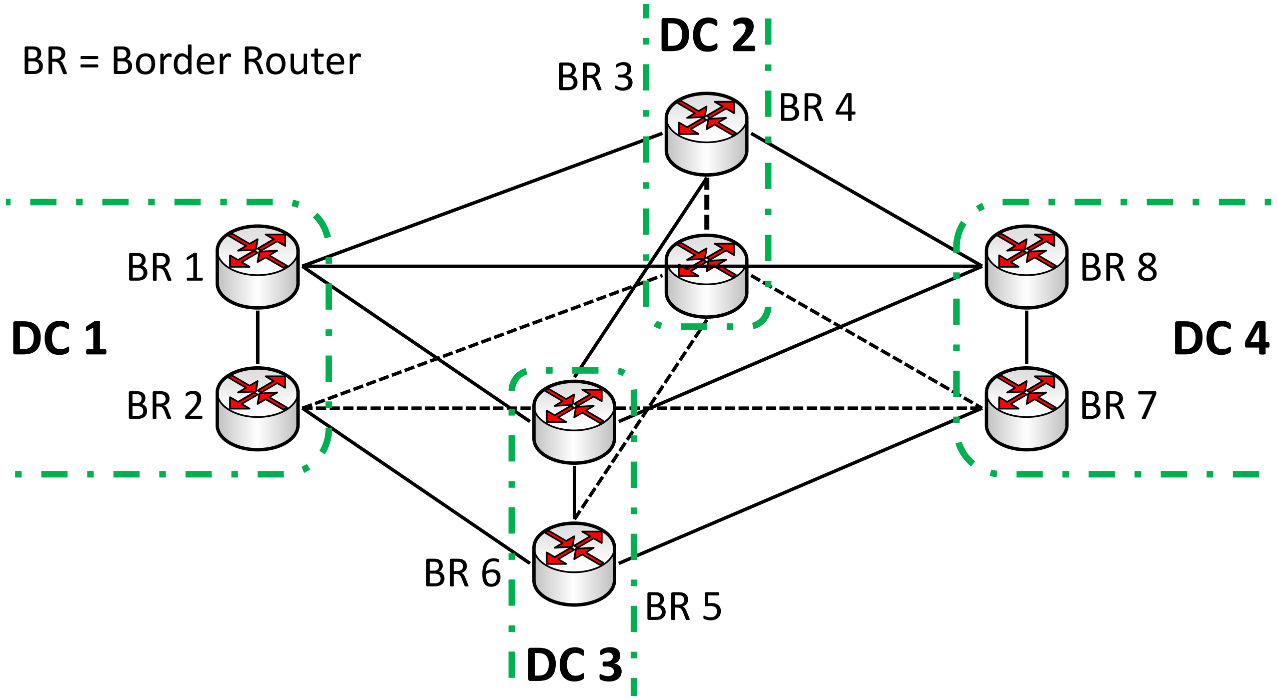
use case: resolving conflicts



solution: statesman coordinates, by locks, swtich_upgrade, TE

- assign TE low-level lock
- switch_upgrade high-level lock

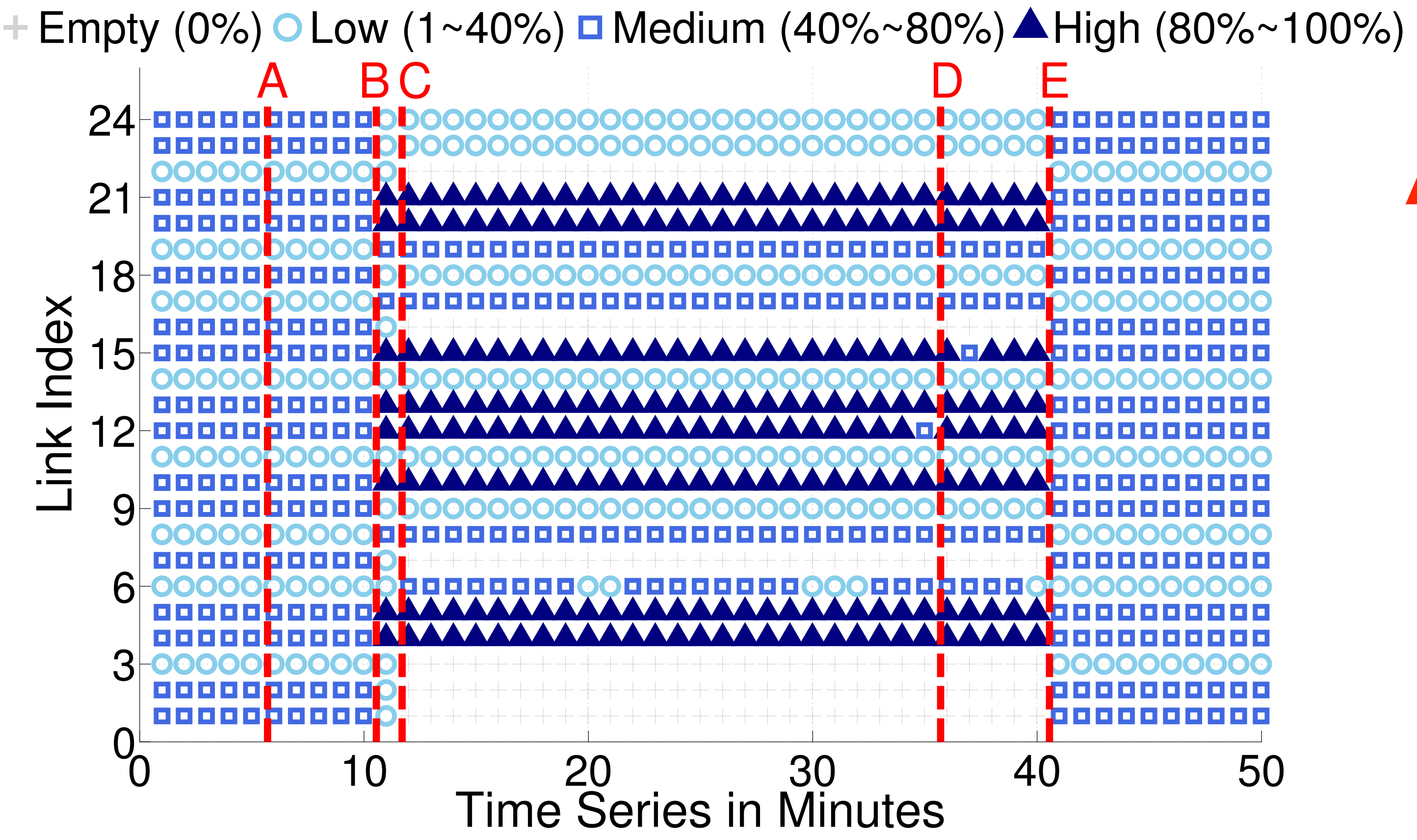
use case: resolving conflicts



statesman coordinates

swtich_upgrade, TE

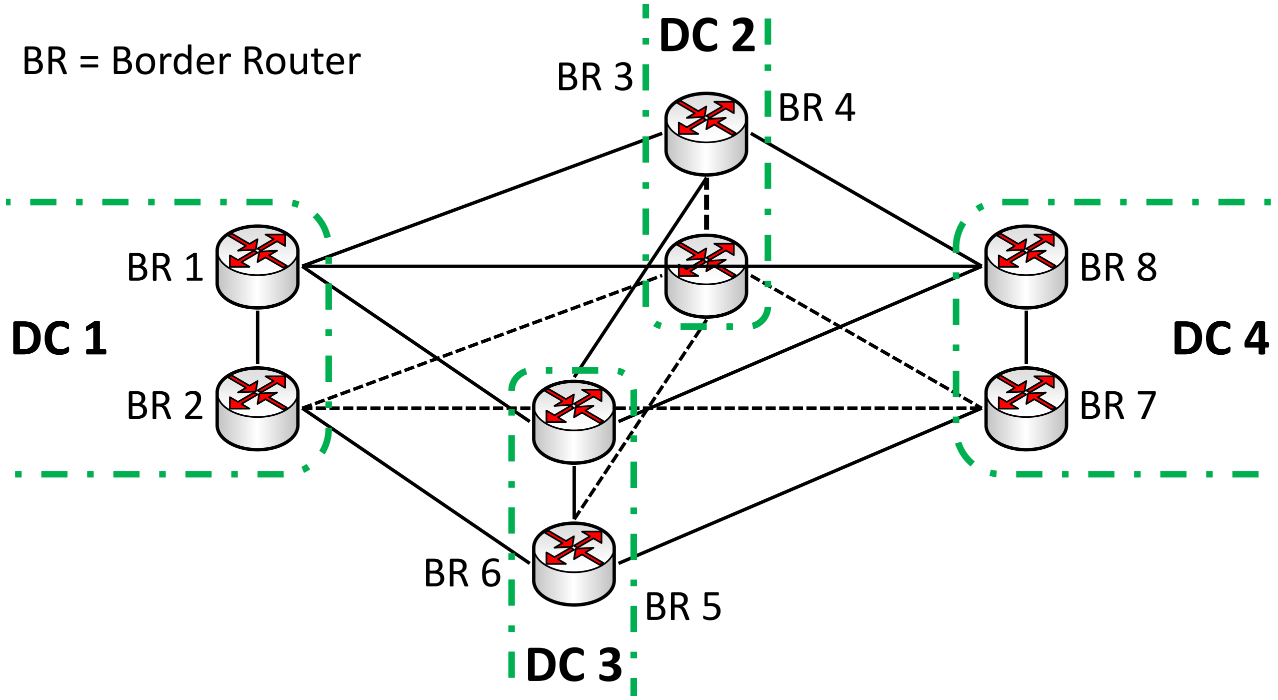
- assign TE low-level lock
- switch_upgrade high-level lock



A

- switch_upgrade acquires high-level lock on BR$_1$

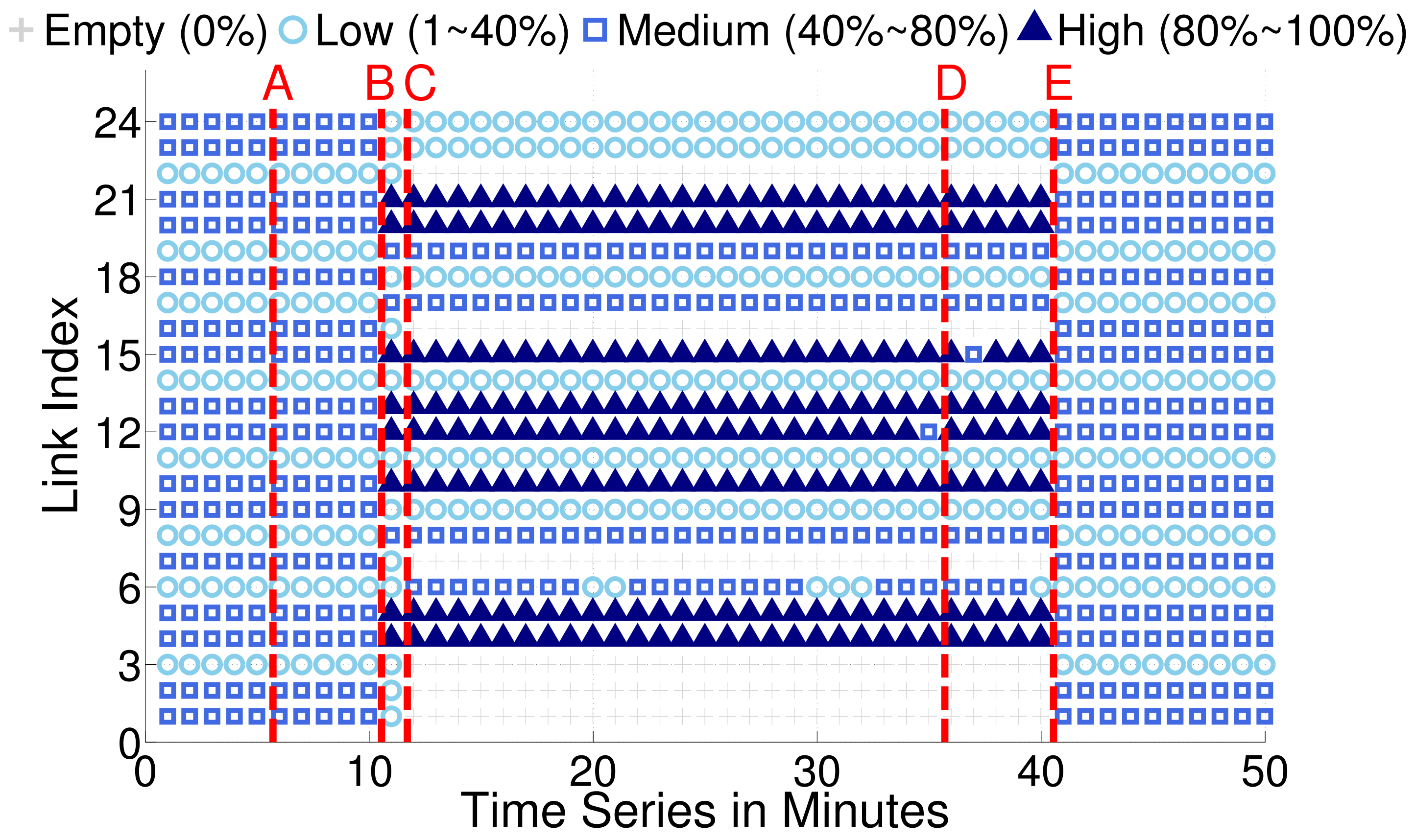
use case: resolving conflicts



statesman coordinates

swtich_upgrade, TE

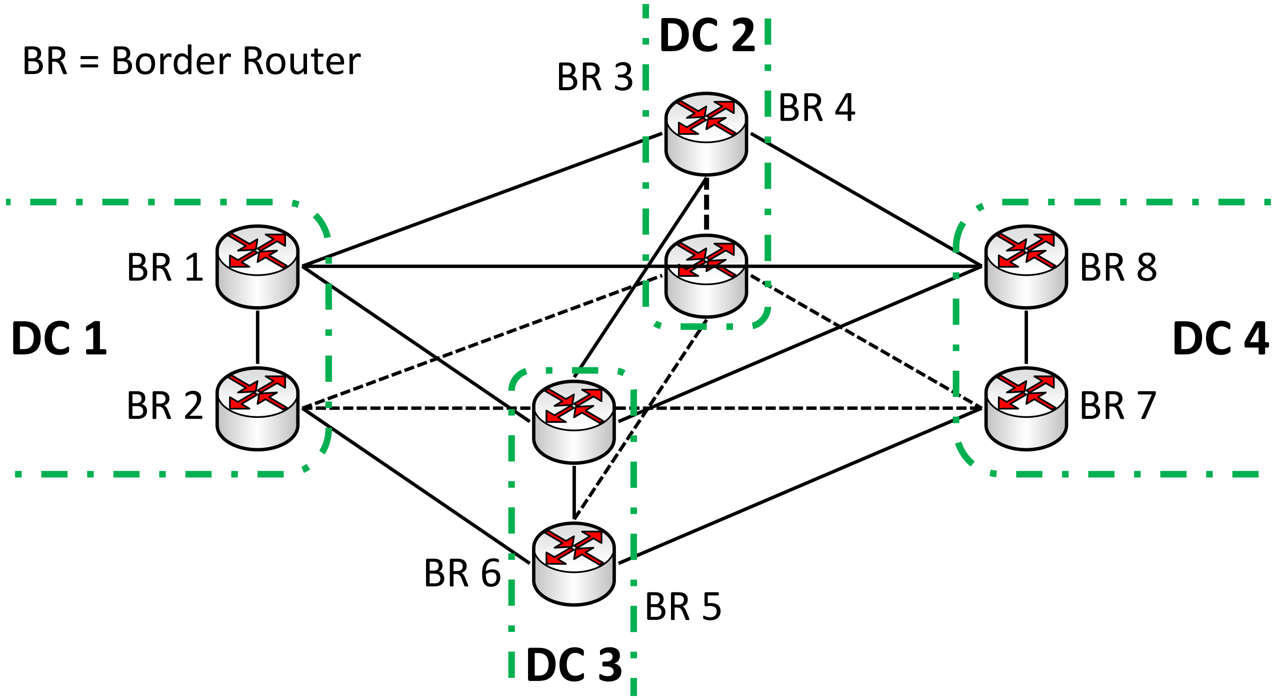
- assign TE low-level lock
- switch_upgrade high-level lock



B

- TE fails to hold low-level lock, moving traffic away from BR$_1$

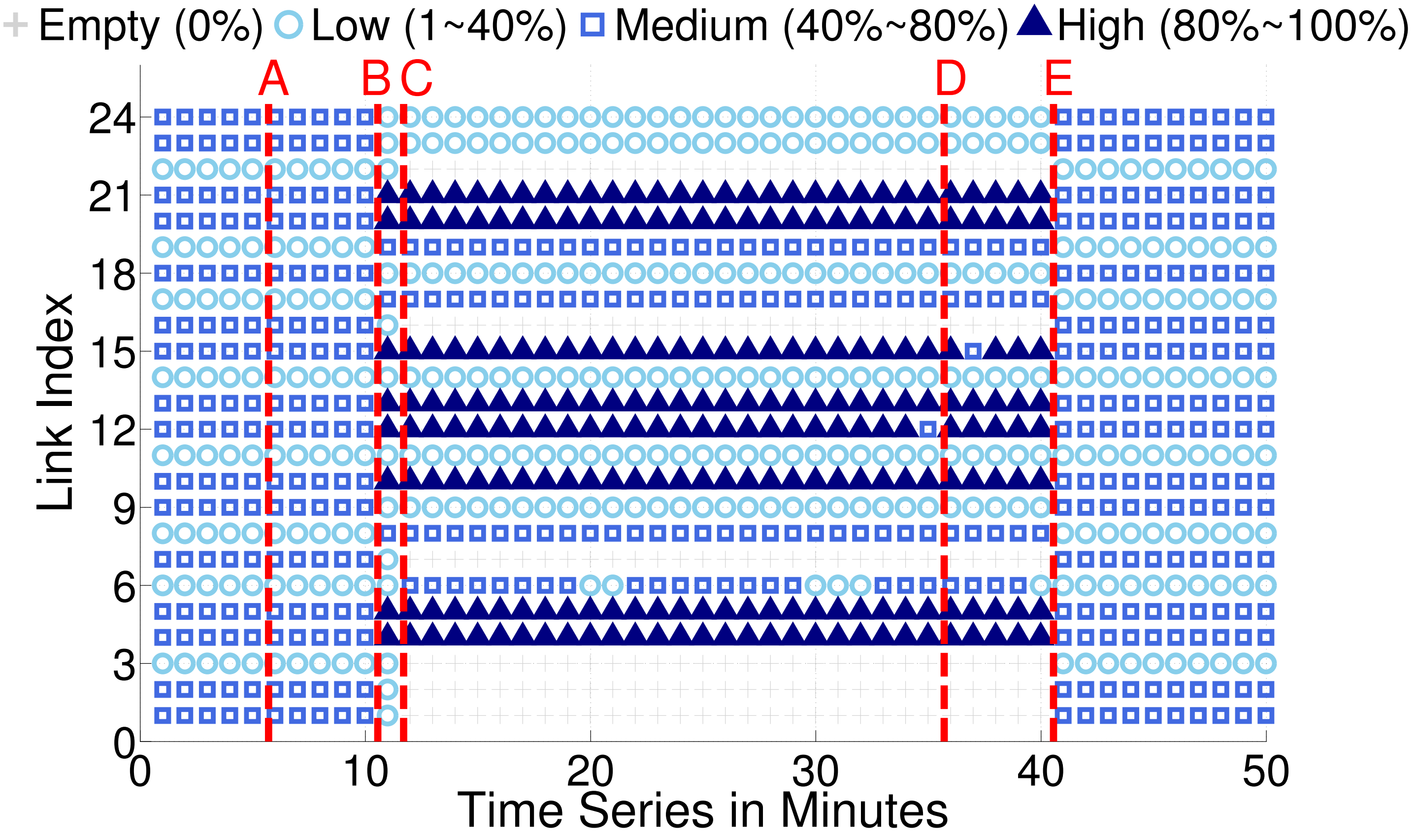
use case: resolving conflicts



statesman coordinates

swtich_upgrade, TE

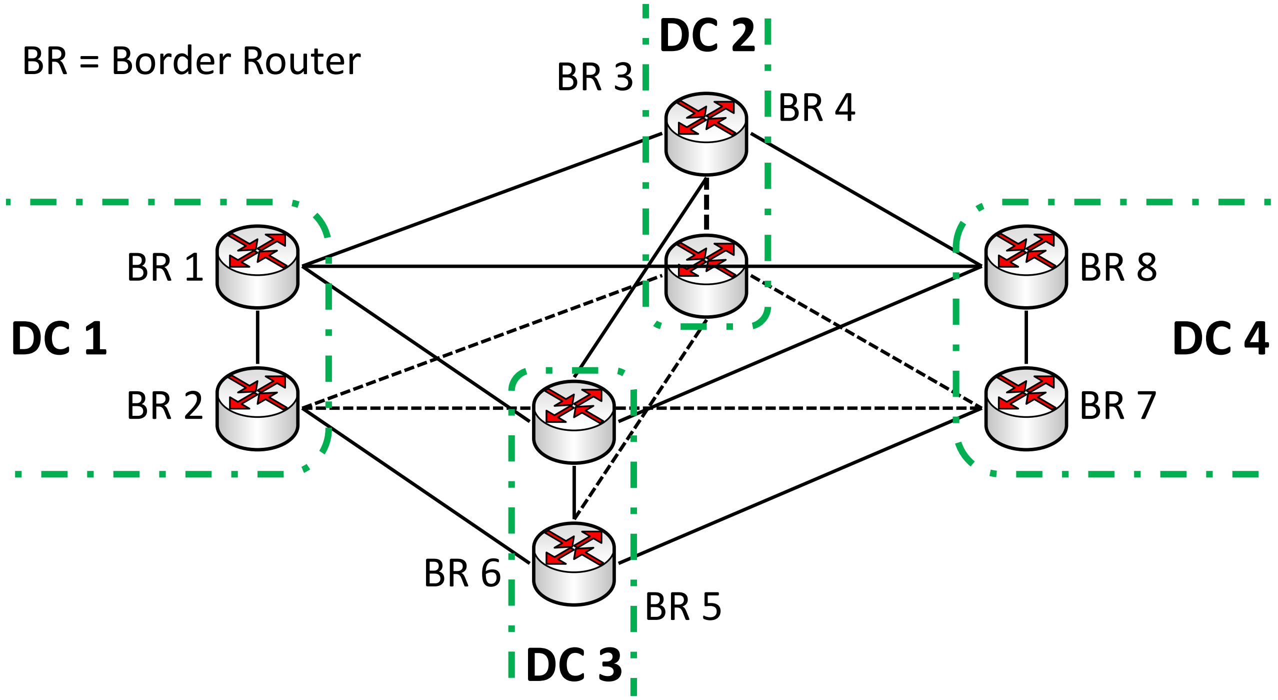
- assign TE low-level lock
- switch_upgrade high-level lock

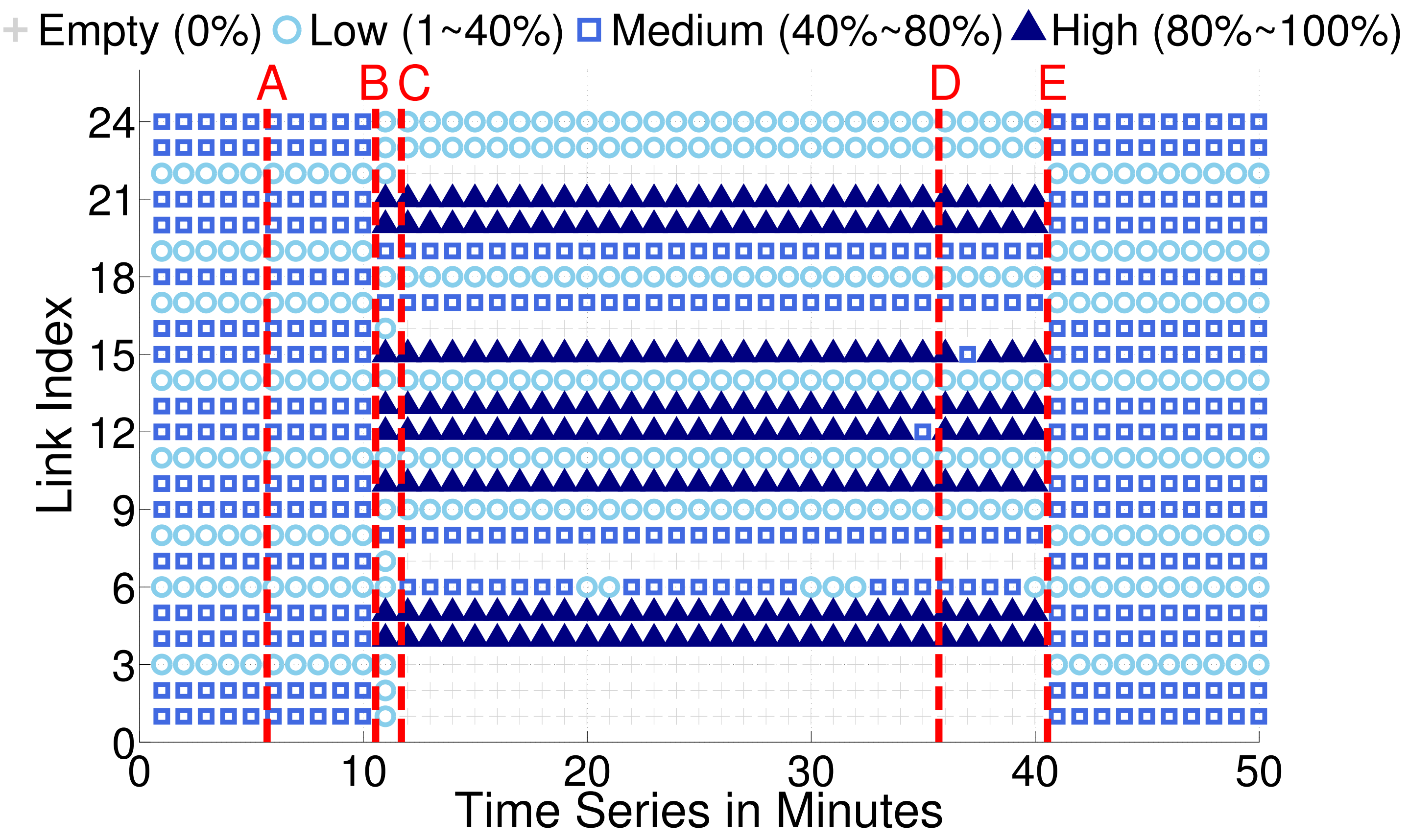


C,D

- C upgrading BR$_1$ in progress
- D upgrading done at BR$_1$, releasing high-level lock

use case: resolving conflicts





statesman coordinates

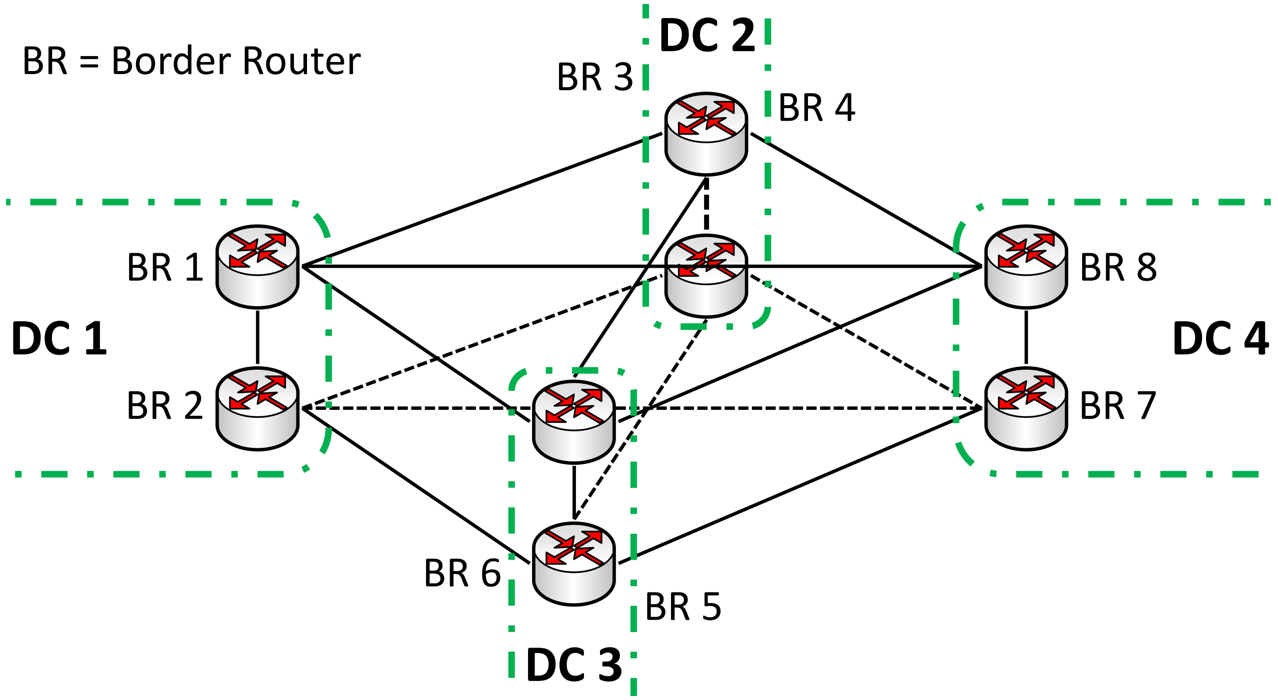
swtich_upgrade, TE

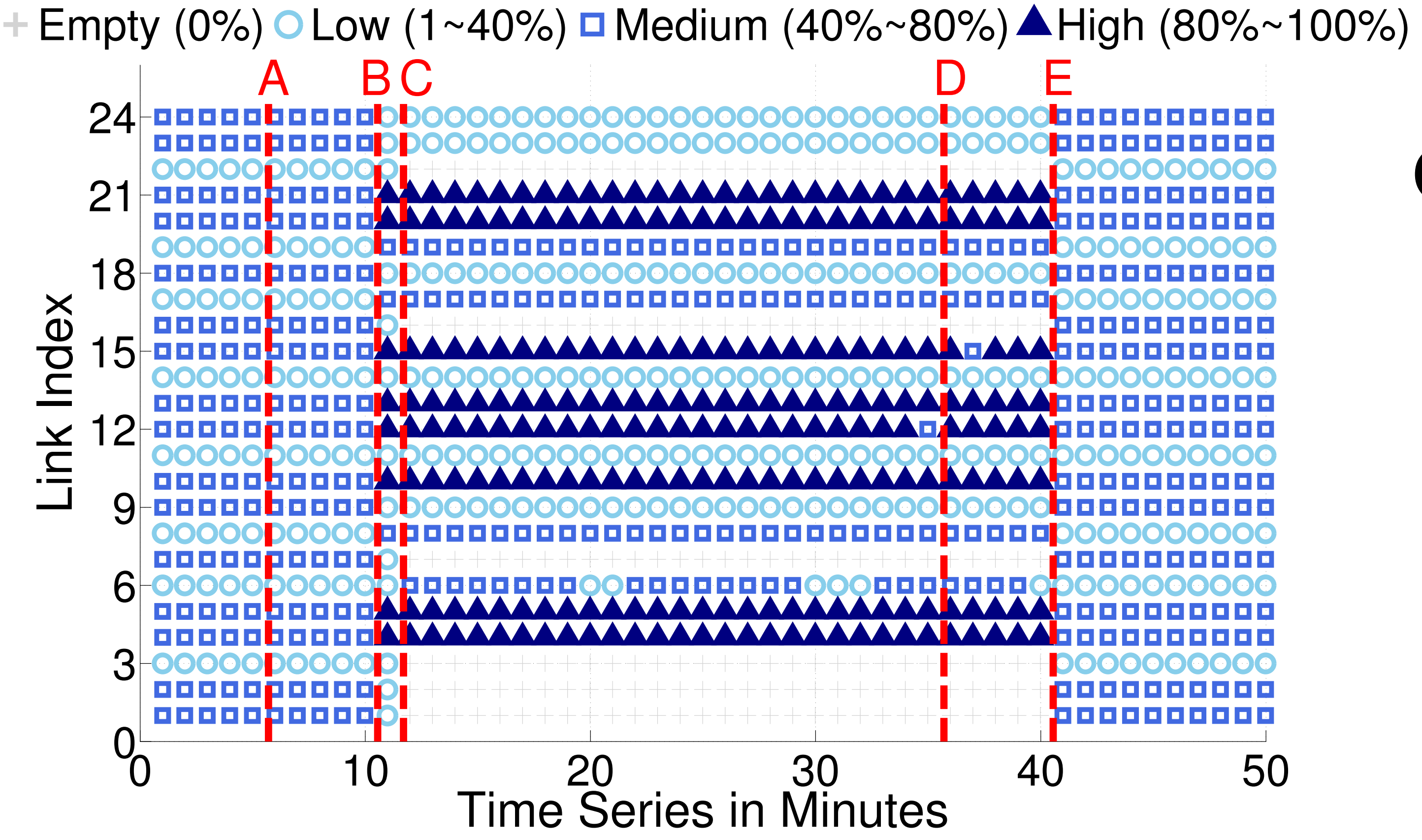
- assign TE low-level lock
- switch_upgrade high-level lock

E

- TE grabs low-level lock, in operation

use case: resolving conflicts





statesman coordinates

swtich_upgrade, TE

- assign TE low-level lock
- switch_upgrade high-level lock

question

- what next?

statesman performance

evaluating latency

- application: (<10ms) negligible
- checker: seconds
- updater: (>50%) dominating