Robustness of Interconnection Networks

3rd JLESC Summer School

Atsushi Hori
RIKEN AICS

Self Introduction

Atsushi Hori - System Software Researcher

The oldest and largest governmental research institute in Japan, since 1917

Advanced Institute for Computational Science (AICS), since 2010

Running the K computer, the largest super computer in Japan (ranked 5th in Top500, Jun., 2016)

involved in the Flagship2020 project
to develop the post-K computer

Self Introduction

Atsushi Hori - System Software Researcher

The oldest and largest governmental research institute in Japan, since 1917

Advanced Institute for Computational Science (AICS), since 2010

Running the K computer, the largest super computer in Japan (ranked 5th in Top500, Jun., 2016)

involved in the Flagship2020 project
to develop the post-K computer

Self Introduction

Atsushi Hori - System Software Researcher

The oldest and largest governmental research

010

DISCLAIMER

This contents of this talk are based on my personal experiences and independent from the Flagship2020 project

京
K computer

Self Introduction

Atsushi Hori - System Software Researcher

The oldest and largest governmental research institute in Japan, since 1917

Advanced Institute for Computational Science (AICS), since 2010

Run[illegible]est super com[illegible]00, Jun., 2016)

inv[illegible]ect to develop the post-K computer

The colored slides are supplements

Self Introduction

Atsushi [illegible] earcher

[illegible] earch

The venue of the next JLESC, in Dec., Kobe

Advanced Institute for Computational Science (AICS), since 2010

Running the K computer, the largest super computer in Japan (ranked 5th in Top500, Jun., 2016)

HPC Network

- Low latency and high bandwidth
  - Higher performance than silicon disks
- High Bi-section bandwidth
  - Low congestion possibility (hopefully)
- Very Reliable
  - No error, No loss
- Dense (in a computer room)
  - Internet covers the whole earth
- Packet Switching
  - No circuit switching (old telephone network)

Outline

Network Basics

- Topology
- Routing
- Implementation

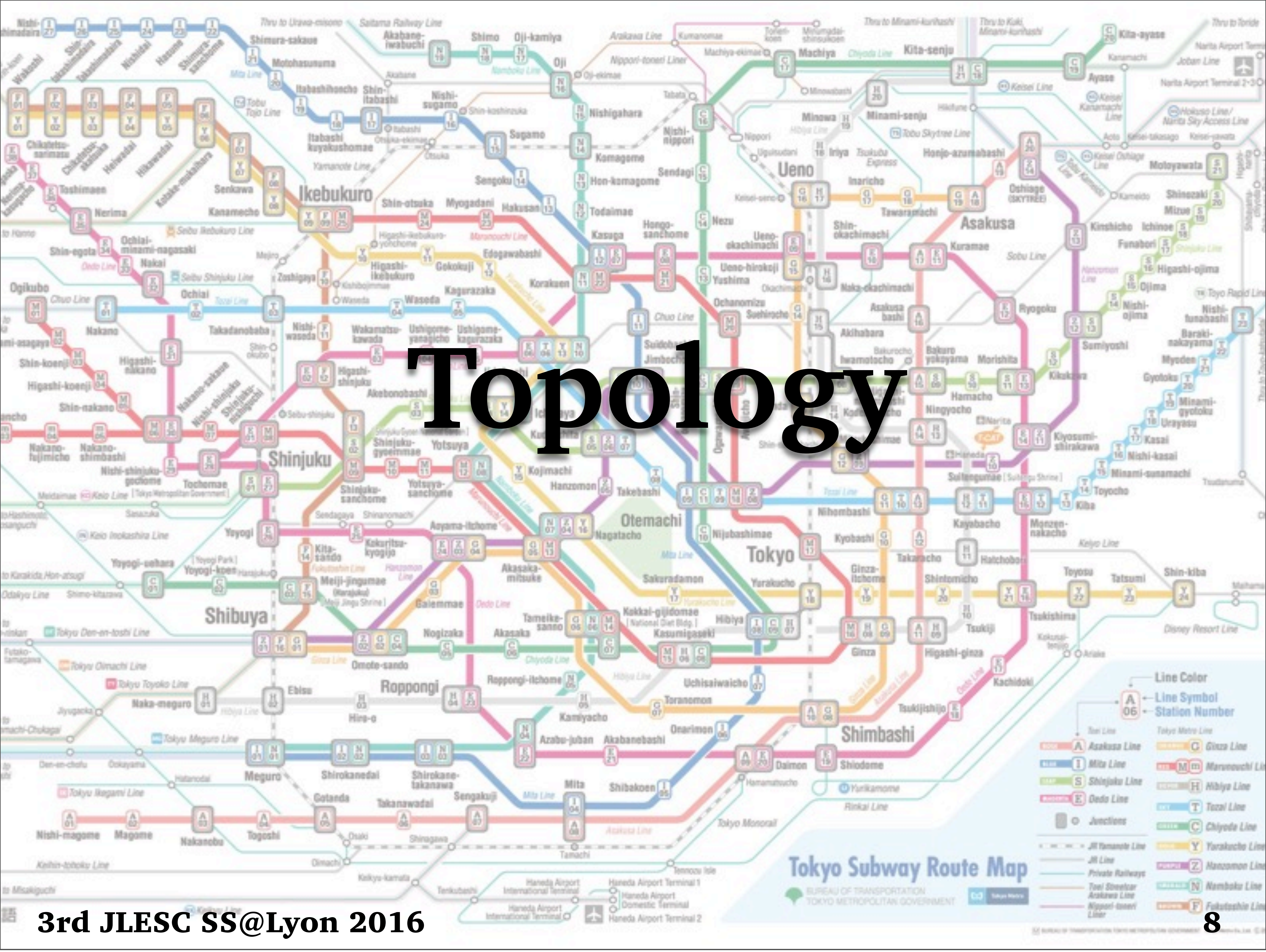
Fault Resilience
+ my personal opinion

Glossary

- A network consists of
  - Nodes where packets are sent and received
    may include a switch (see below)
  - Switches (Routers)
  - Links connecting nodes and switches

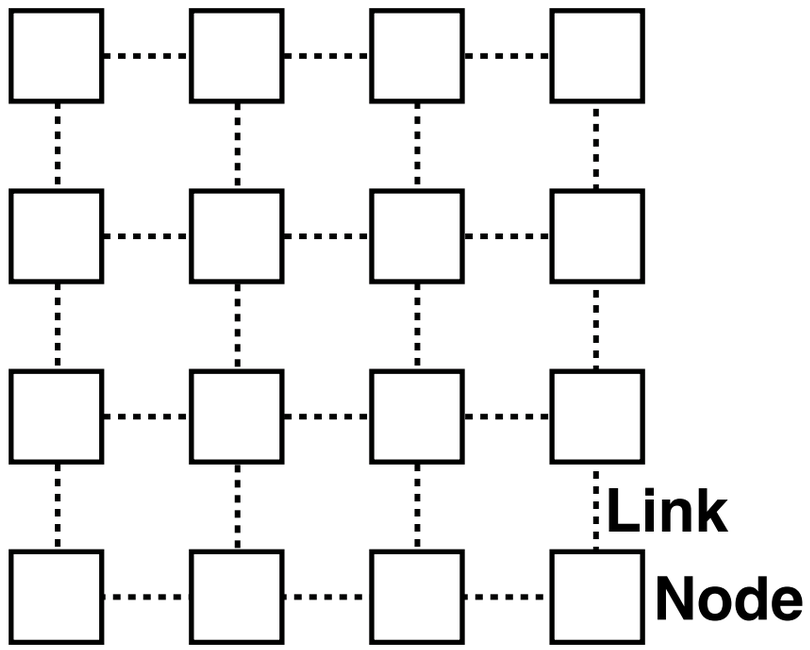
- Data transfer
  - Packet a unit of transfer
  - Message consists of multiple packets

Topology

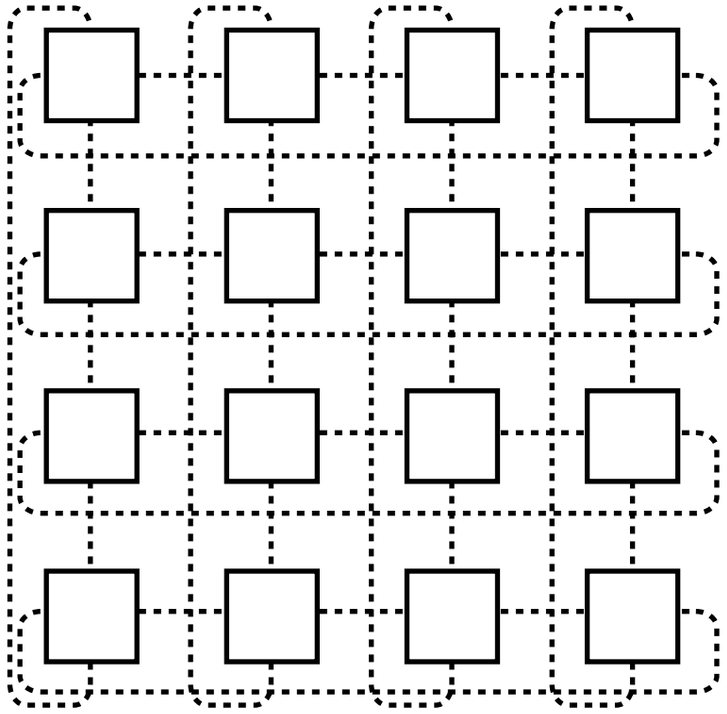

Tokyo Subway Route Map
Ikebukuro
Ueno
Asakusa
Shinjuku
Otemachi
Tokyo
Shibuya
Roppongi
Shimbashi
Line Color
Line Symbol
Station Number

Network Topologies (1)

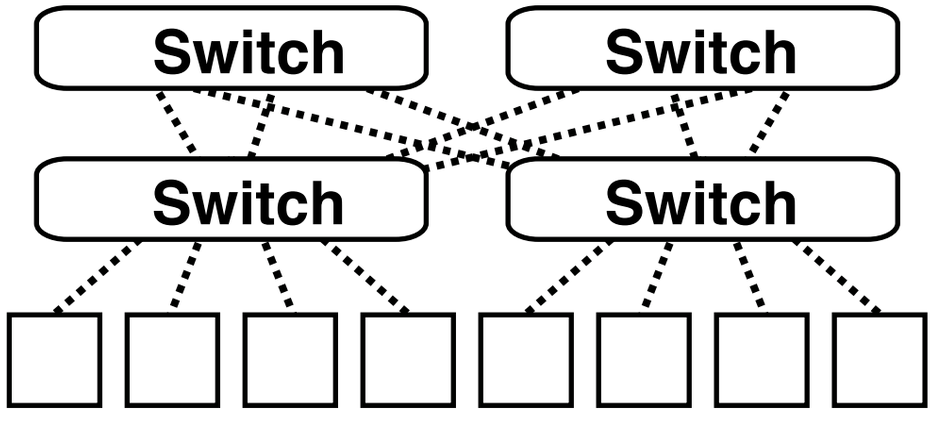
Mesh



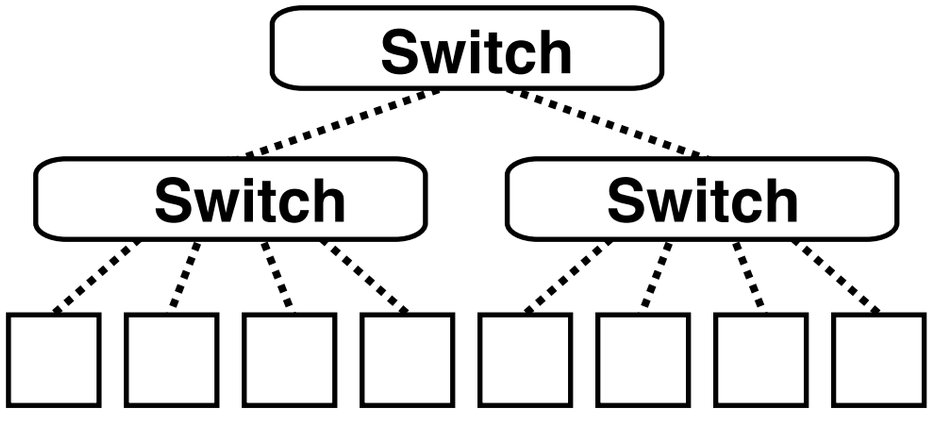
Torus



FatTree

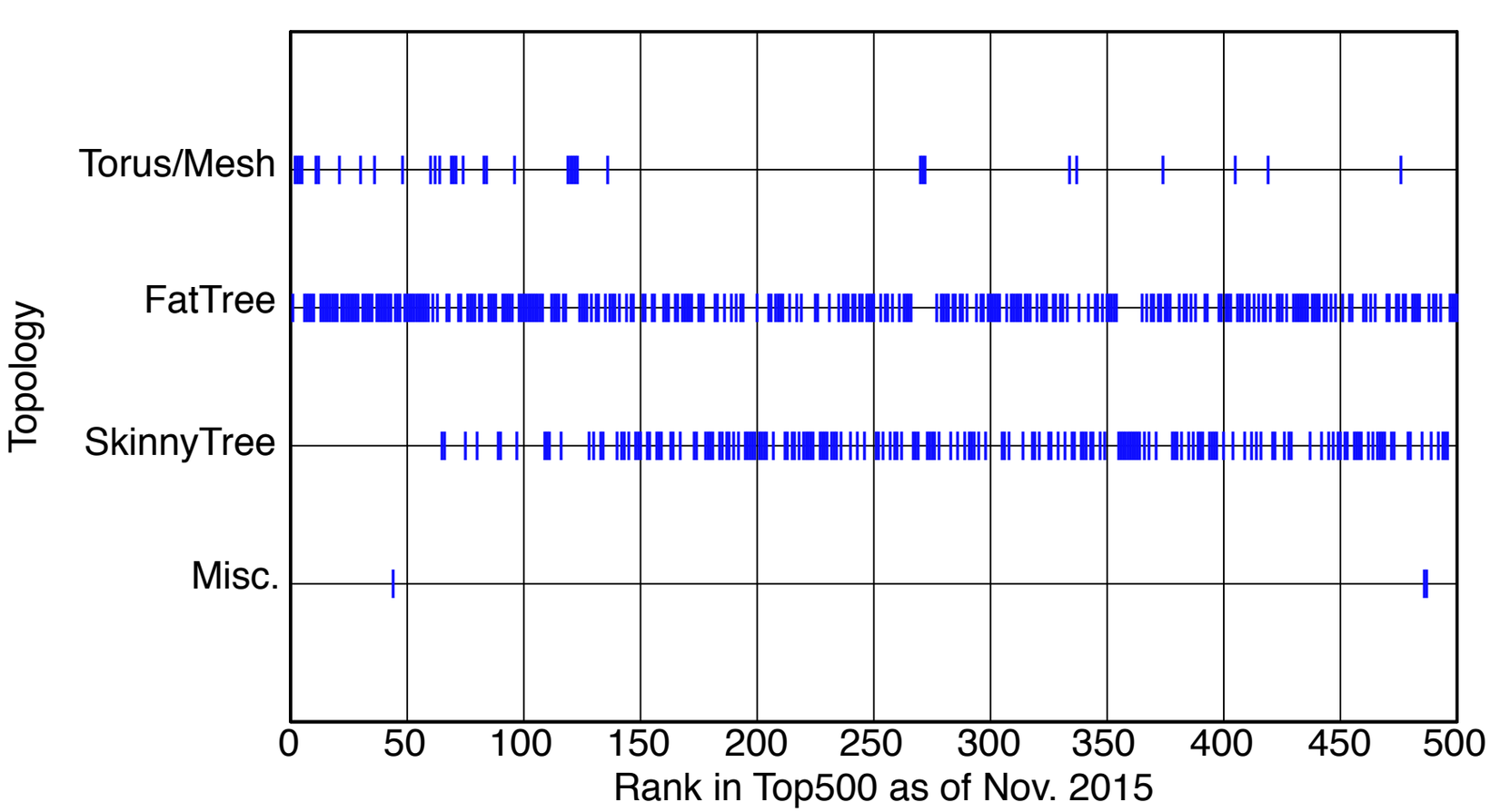


“SkinnyTree”



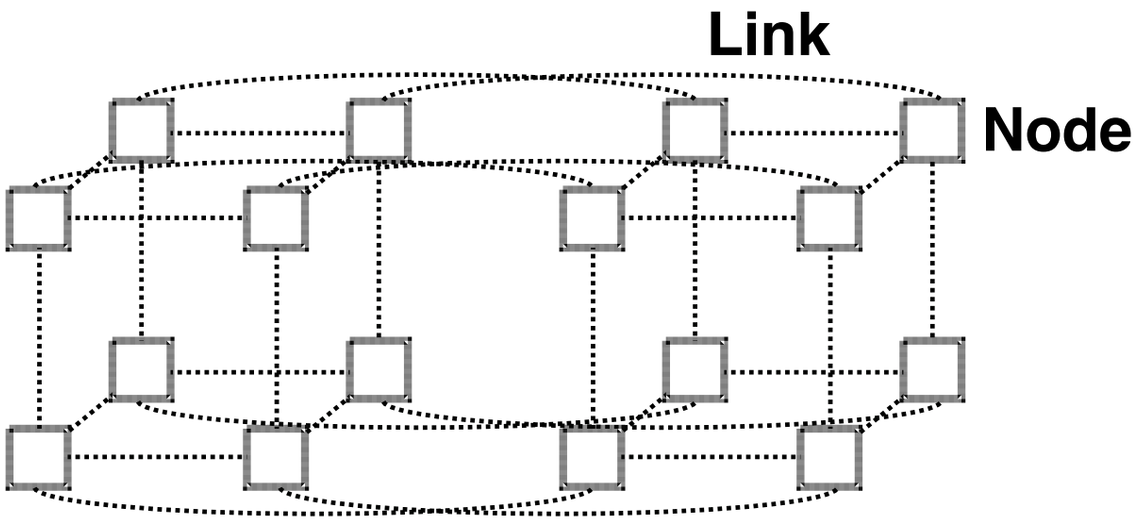
Network Topology in Top500

- Topologies in Top500 http://www.top500.org
  - Torus/Mesh BG/Q, the K (Tofu)
  - FatTree *Infiniband*, *Aries*, Cray Gemini, Tiahne
  - SkinnyTree *Ethernet*
  - Misc. IBM Power 775

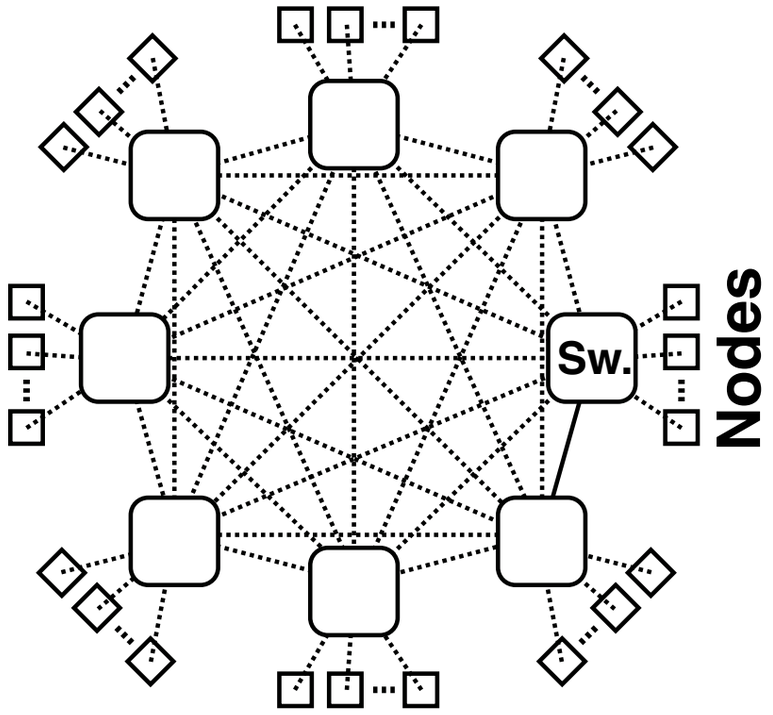


Network Topologies (2)

Hypercube



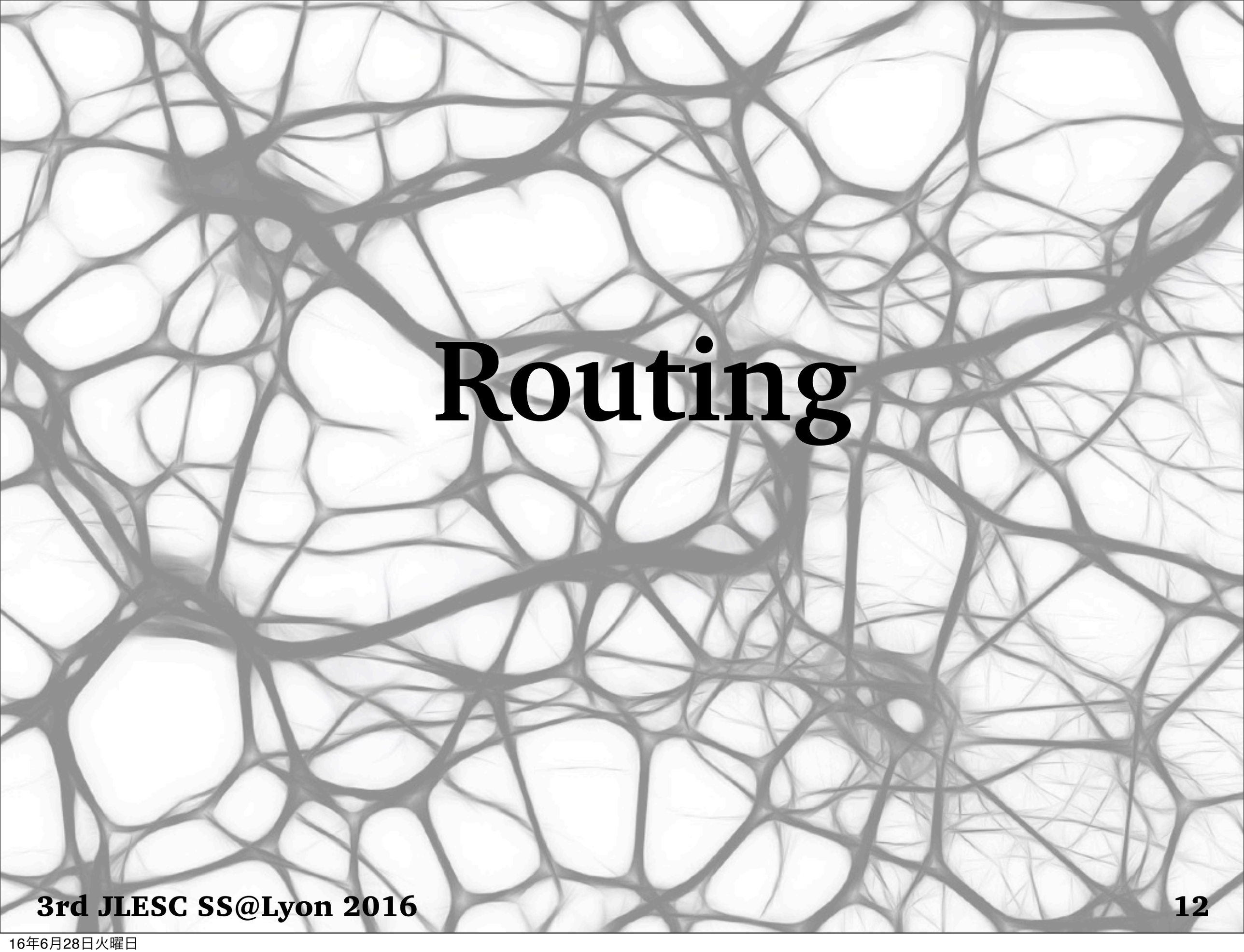
Dragonfly



CM-2, nCUBE in 90s

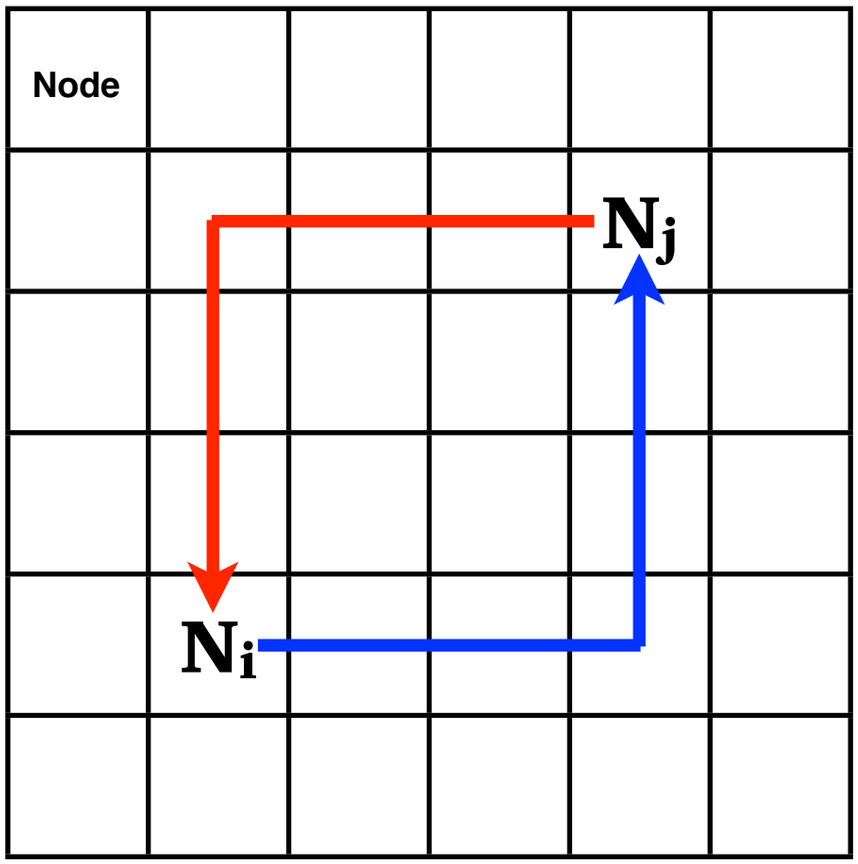
Cray XC series

and many others (*ring*, *star*, *butterfly*, to name a few)


Routing

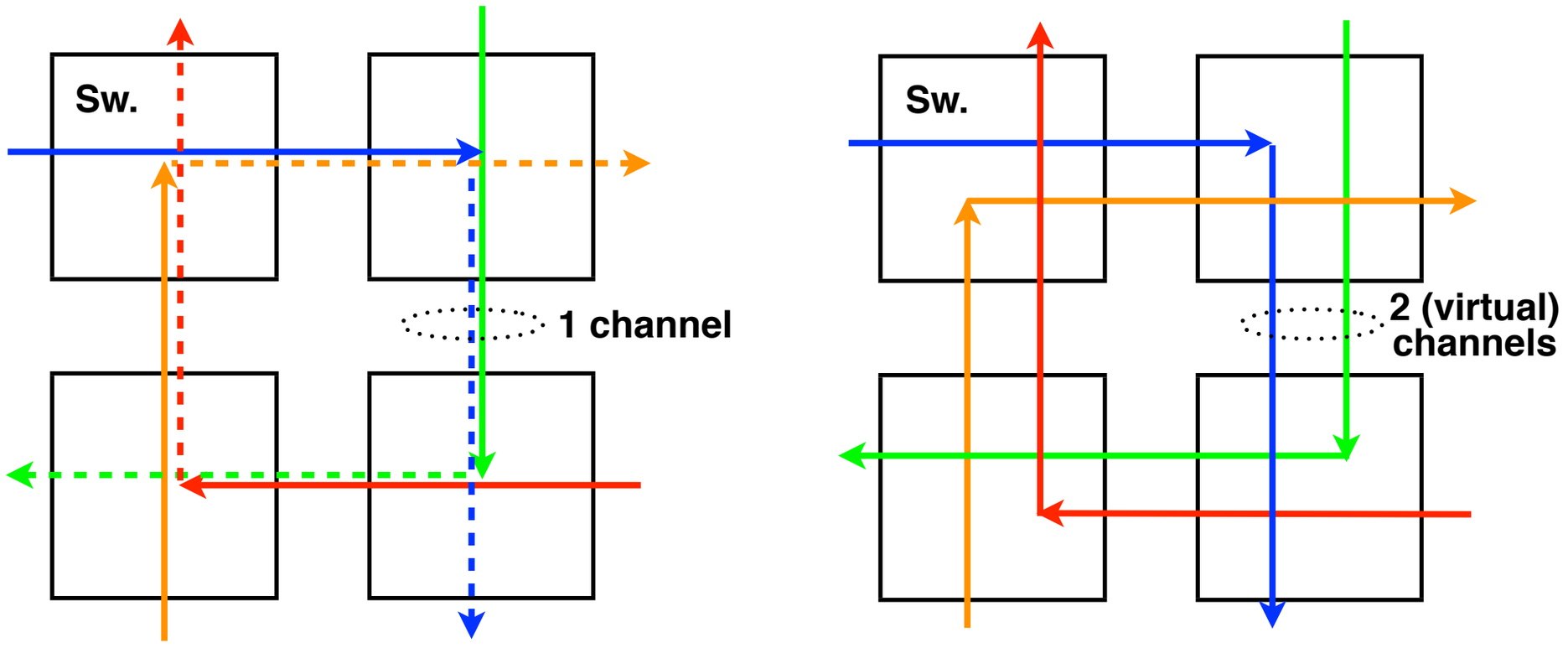
Routing

- Find a path from a sender node to a receiver node
- Ex) X-Y (Dimension Order) Routing in 2D Mesh



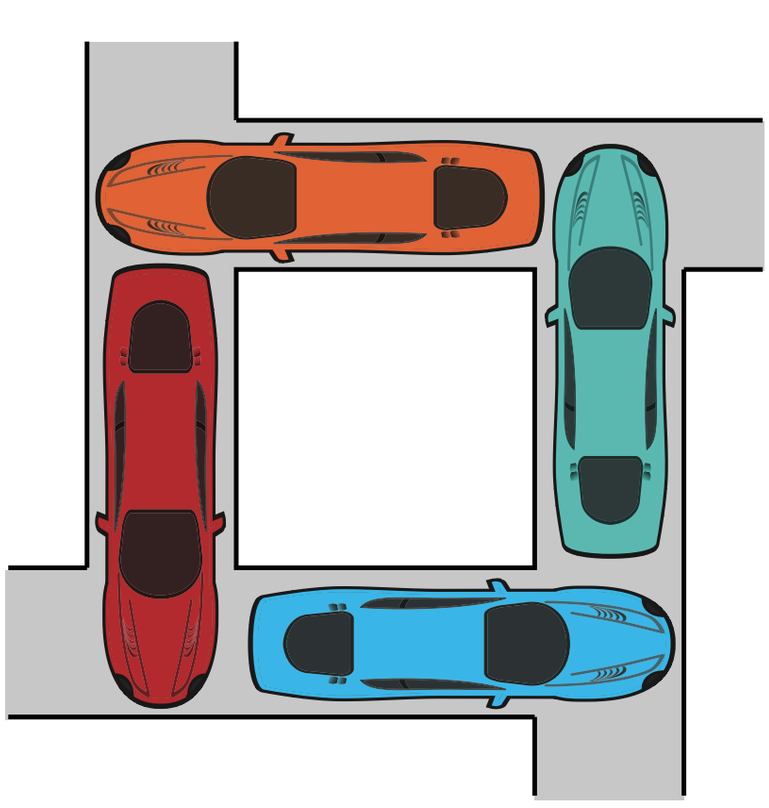
Deadlock

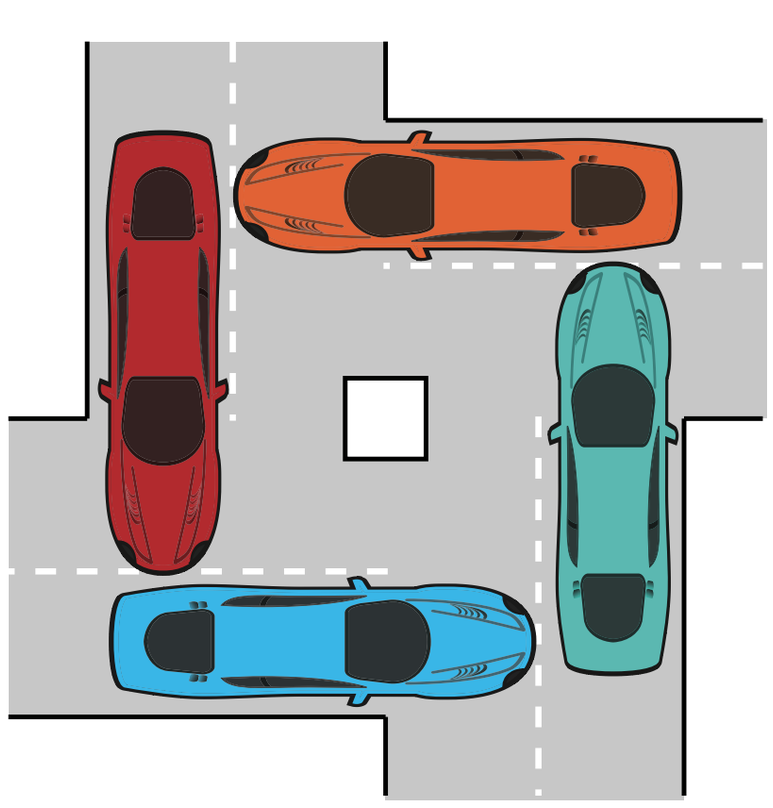
- A routing algorithm on a network topology must be deadlock free
  - Cyclic path can cause deadlock
  - Deadlock can be avoided by having bypass
    - Virtual channels



Deadlock

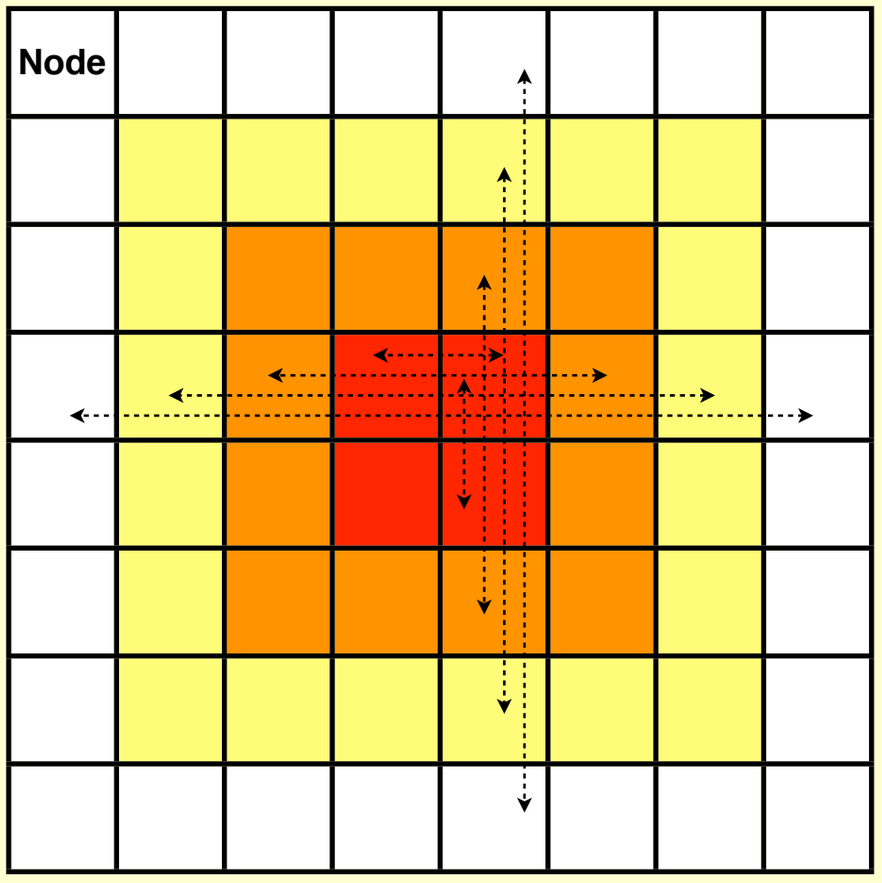
- A routing algorithm on a network topology must be deadlock free
  - Cyclic path can cause deadlock
  - Deadlock can be avoided by having bypass
    - Virtual channels





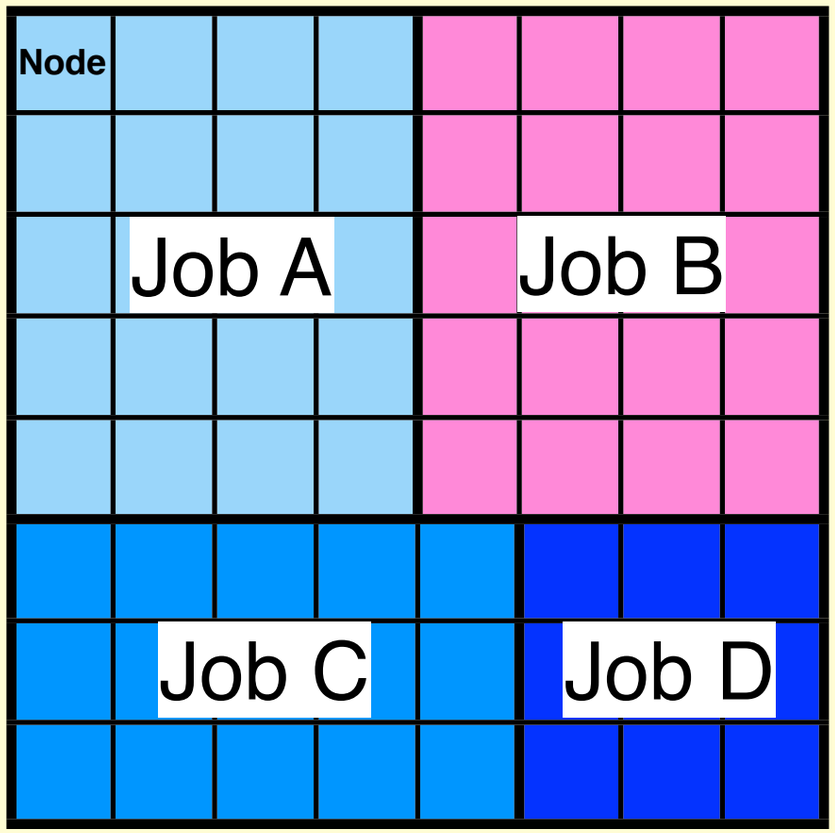
Hot Spot

- Hot spot
  - Packet congestion happens
- 2D Mesh　Hot spot at the center
- 2D Torus　No hot spots

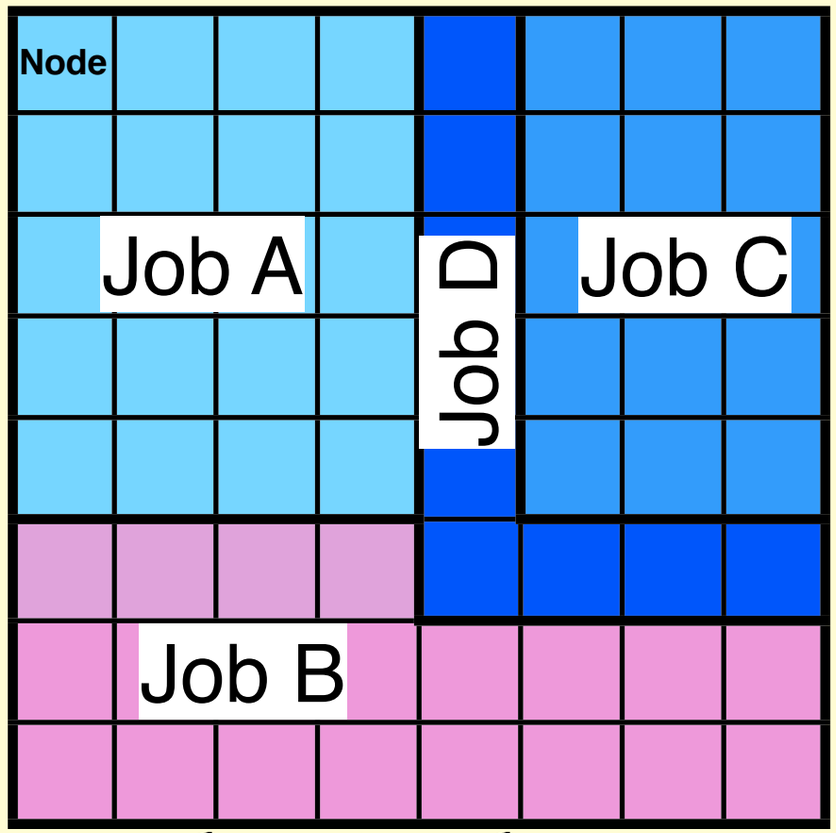


Partitioning

- Multiple jobs can run on a big machine
- Node space is partitioned
  - Partitioning may change topology of a job
  - Jobs may have interference



2D torus turns into 2D mesh

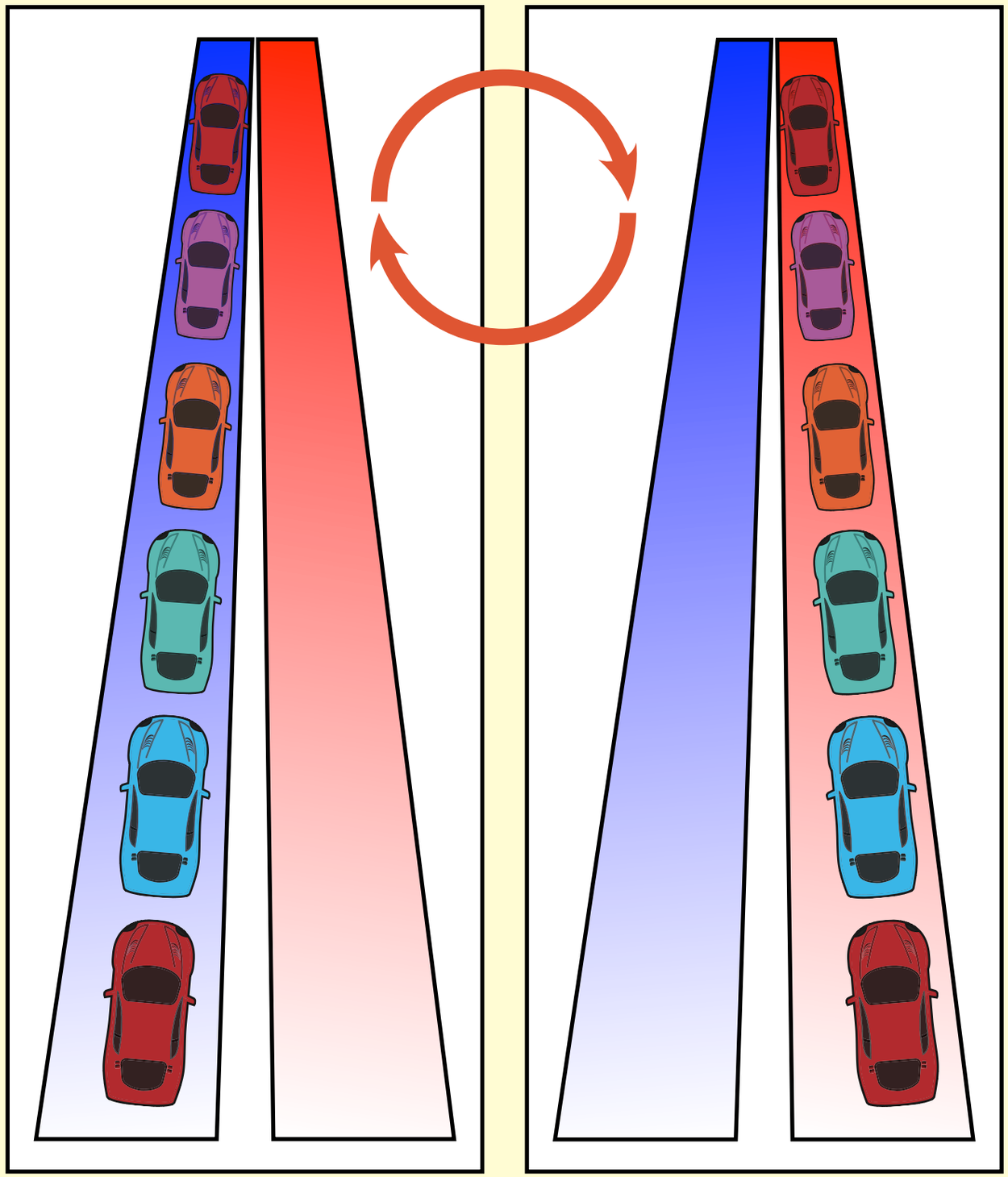


Job B, C and D can interfere with the others

Dynamic (Adaptive) Routing

- Static Routing
  - Once a path is fixed, packets go along with the path
- Dynamic (adaptive) Routing
  - Paths can be changed dynamically according to the state of the network
- Issues
  - Algorithm: how, who, when ?
  - Deadlock free
  - Route changing latency & H/W resource
  - Stability (see next slide)
  - Packet order is not preserved (see next of next)

Oscillation in Adaptive Routing



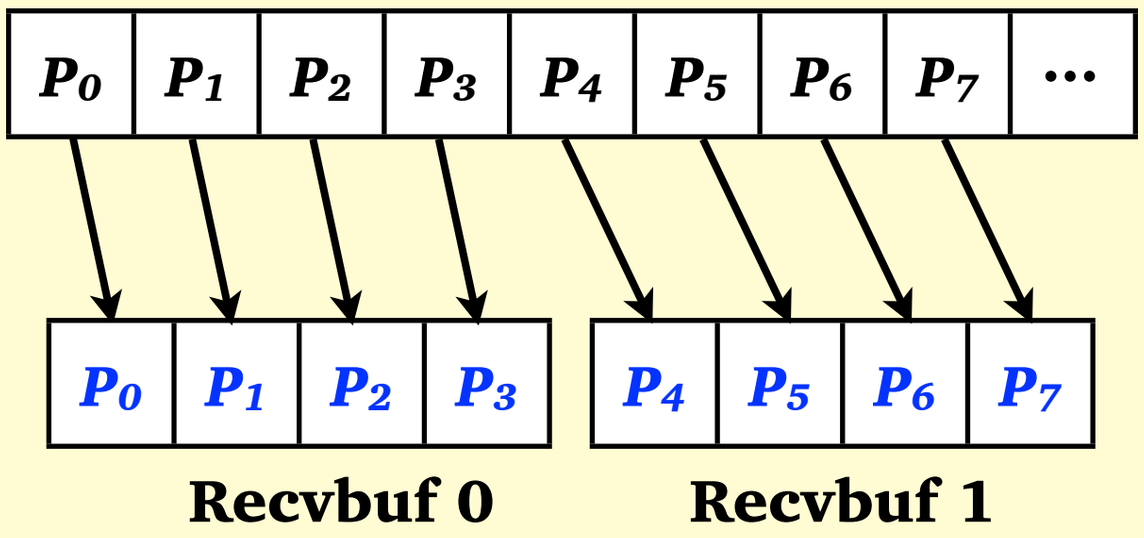
Two roads to the same destination

1. One is very crowded
2. The radio says the other is empty
3. Everybody rushes into the other road
4. (repeat 1-3)

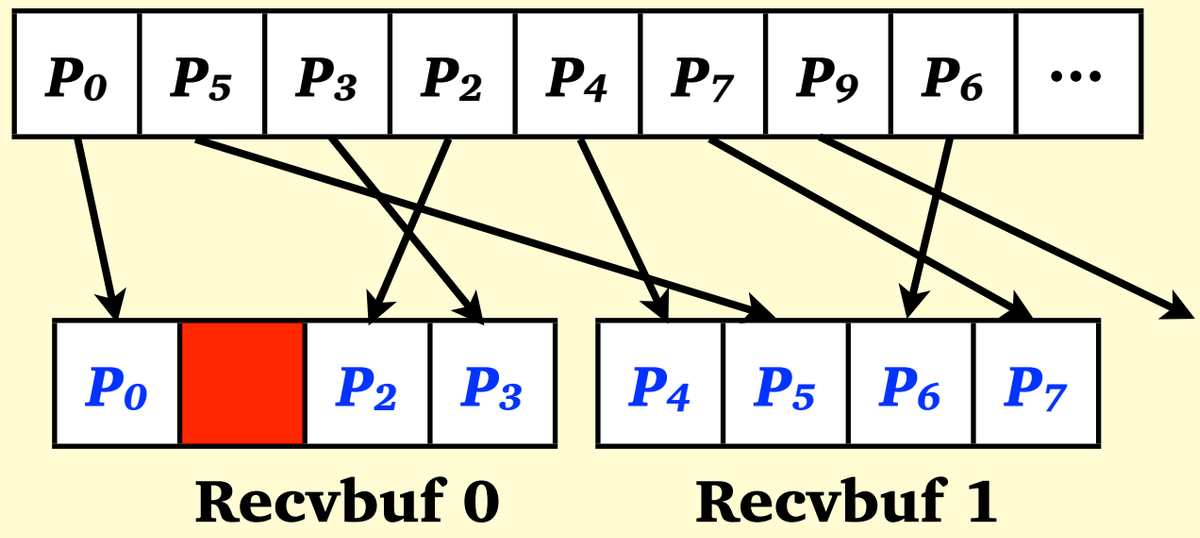
Packet Order

- Adaptive routing cannot preserve packet ordering
- This can be problematic when receiving large messages consisting of multiple packets

Sending Order = Receiving Order



Sending Order ≠ Receiving Order



Metrics

- Topology
  - The higher radix, the smaller network diameter
    - Network Diameter
    - High-Radix or Low-Radix
- Performance
  - Whole
    - Bisection Bandwidth
  - P2P
    - Bandwidth and Latency
    - Hop count
  - Collective Operations (Barrier, and so on)
    - Latency

Implementation

Installation of the K Computer

Direct/Indirect Network

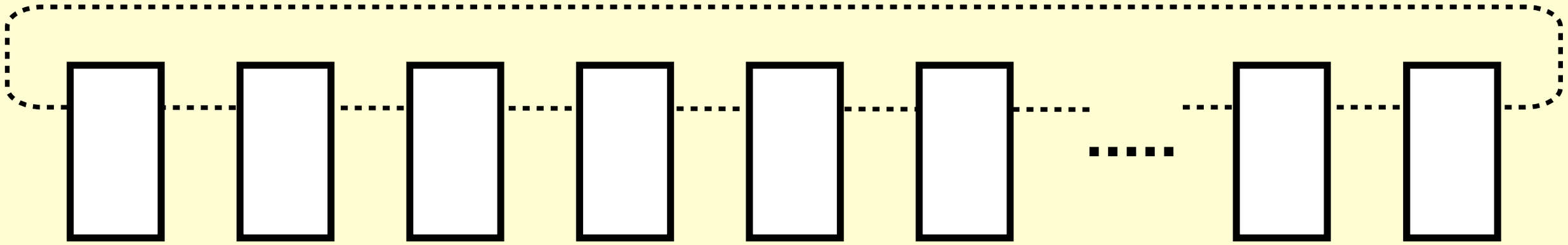
- Direct or Indirect Network
  - Direct network
    - Every node has a switch inside
  - Indirect network
    - Node has no switch

Note: In many books, direct or indirect network is categorized as an aspect of topology

Machine/*Network*	Direct/Indirect
the K (*Tofu*)	Direct
BG/Q	Direct
Infiniband	Indirect
Ethernet	Indirect

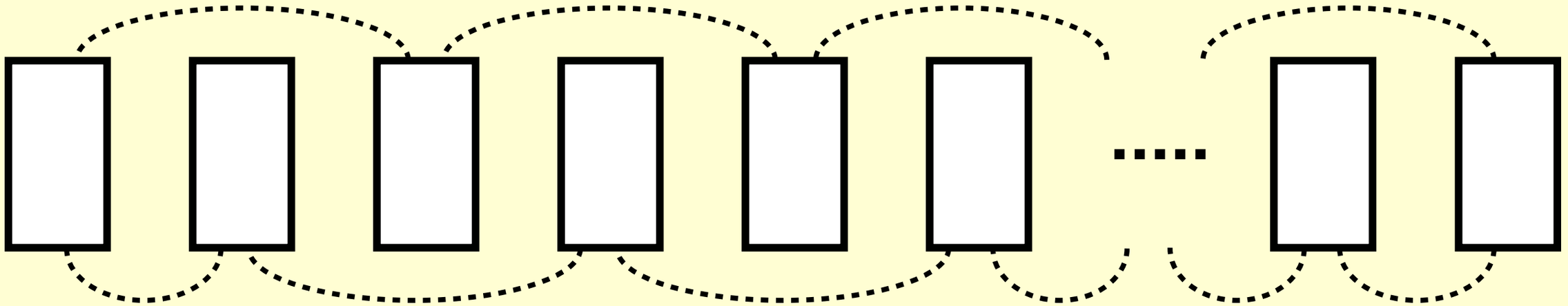
Level off Cable Lengths

- Naive implementation results in uneven cable lengths



Level off Cable Lengths

- To level off cable lengths, alternate nodes are connected



Co-Design

- Network Cost = $\sum C + \sum S + \sum L$

  C: Network interface (card) of a node

  S: Switch

  L: Cable

- Co-design
  - Communication patterns of applications
  - Find protocols to maximize performance of possible applications, and
    - to minimize network cost
    - to minimize power consumption

Fault Resilience

Fault Resilience

- System and/or jobs can survive from a network component failure
  - Possible failure points
    - Link
    - Switch
    - Node

Link or Switch Failure

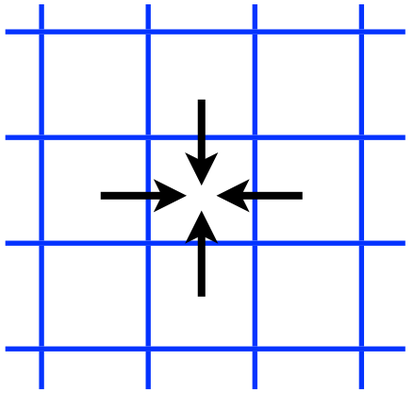
- Static routing
  - *Somebody* changes routing info. to bypass failed part(s)

- Dynamic routing
  - If a failure can be detected, the failed part(s) can be automatically bypassed
  - Needless to say it must be deadlock free

Node Failure

- If application has
  - Dynamic load balancing
    - Job stops using the failed node, and rebalance load
  - Static load balancing
    - Ex) Stencil Computation
      - hard to rebalance load => spare node

2D array V(N,M)

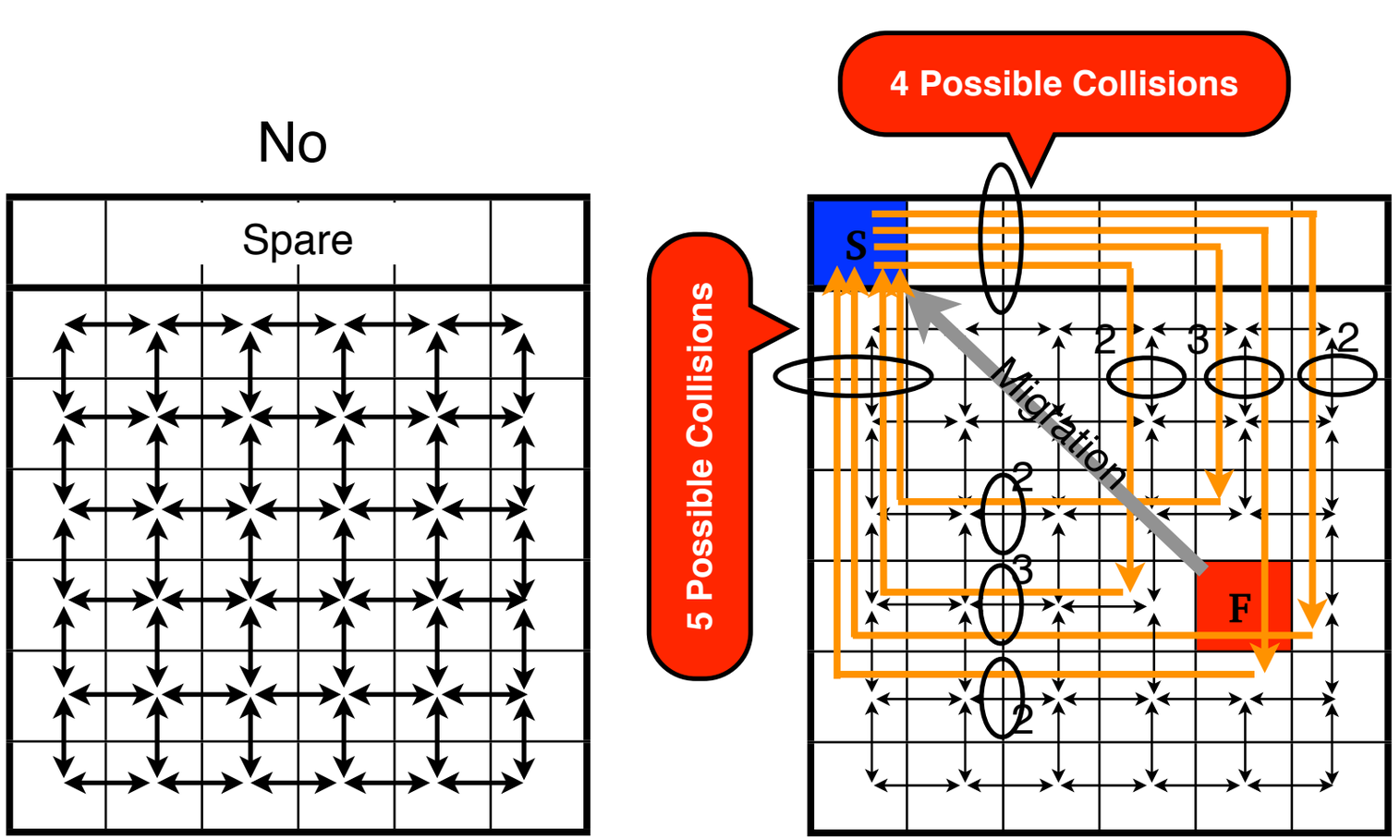


2D Jacobi iteration

V'(i,j) = A * (V(i-1,j) + V(i+1,j) + V(i,j-1) + V(i,j+1))

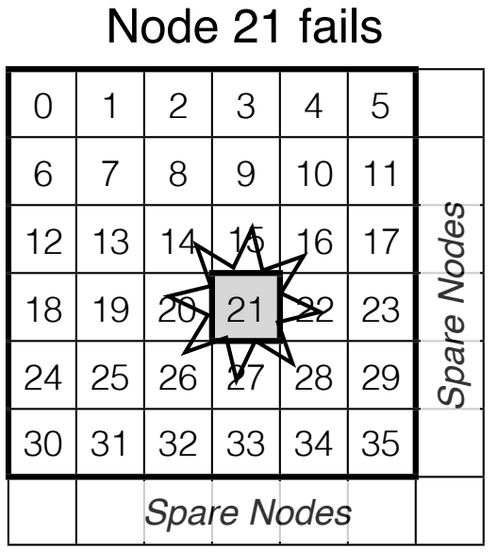
Spare Node Substitution

- Assuming switches and links are all healthy
- A naive spare node substitution may result in a large number of packet collisions
  - Max. latency depends on #collisions
- Is there a way to avoid this situation ?



Spare Node

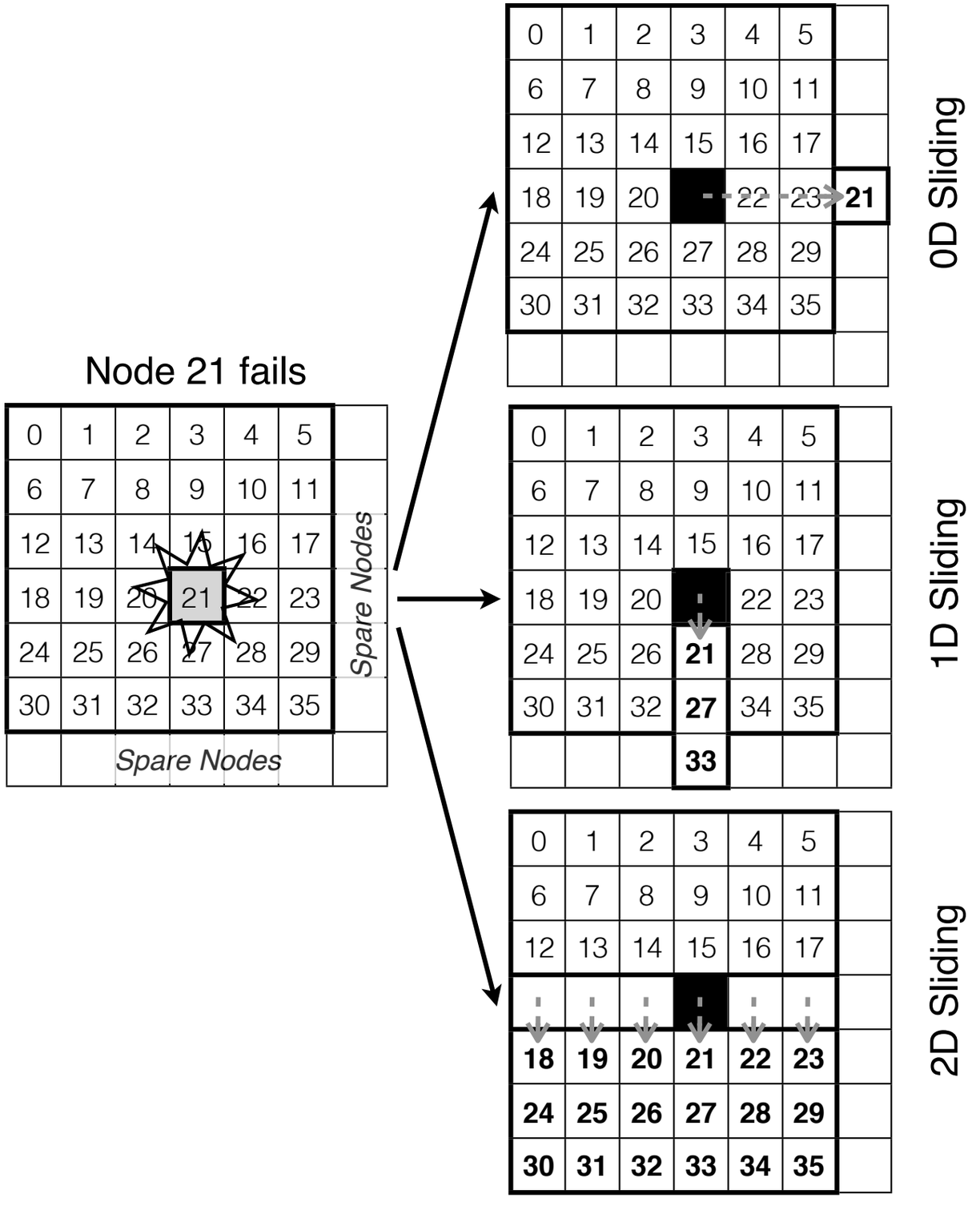
- Pros
  - Easy to program
  - Balanced load
- Cons
  - Lower node utilization
  - Additional packet collisions

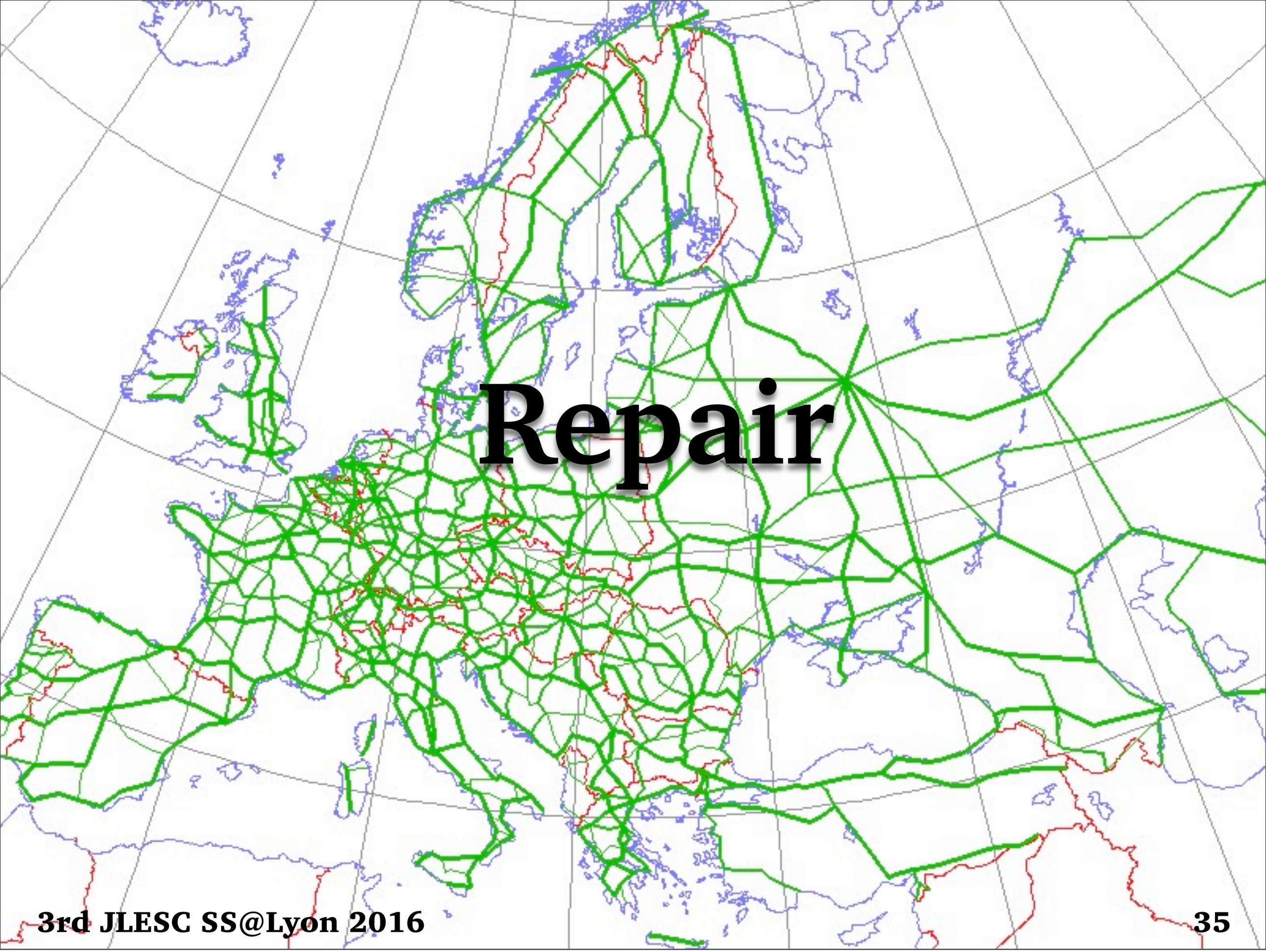


Sliding Methods - Basic Idea

- Sliding Methods
  - 0D - Naive method
  - 1D - Up to 3 (worst case)
  - 2D - Up to 2
  - *and so on*

- Hybrid Sliding
  - Try the highest degree method first
  - If failed, try lower degree method

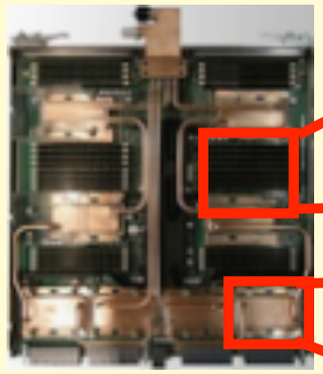



Repair

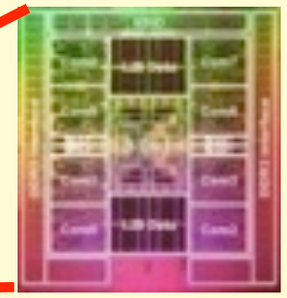
The K and FX100

The K Computer 2011

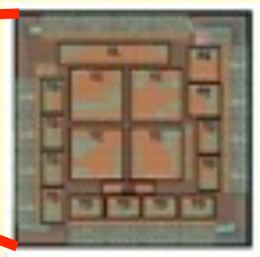
24 chassis (768 cores)



1 system board 4 CPUs and 4 ICCs (32 cores)



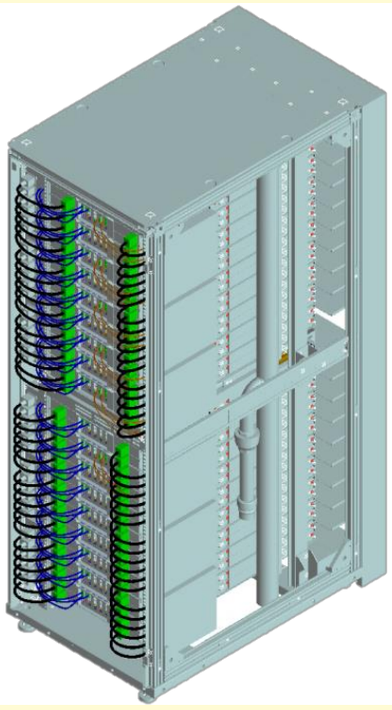
CPU 8 Cores



ICC (Tofu Network)

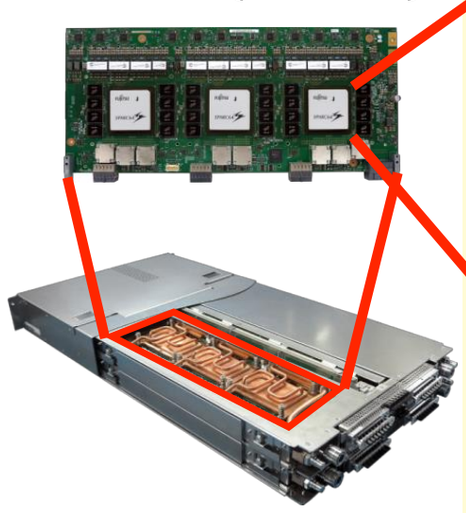
FX100 2015

http://accc.riken.jp/wp-content/uploads/2015/06/chiba.pdf

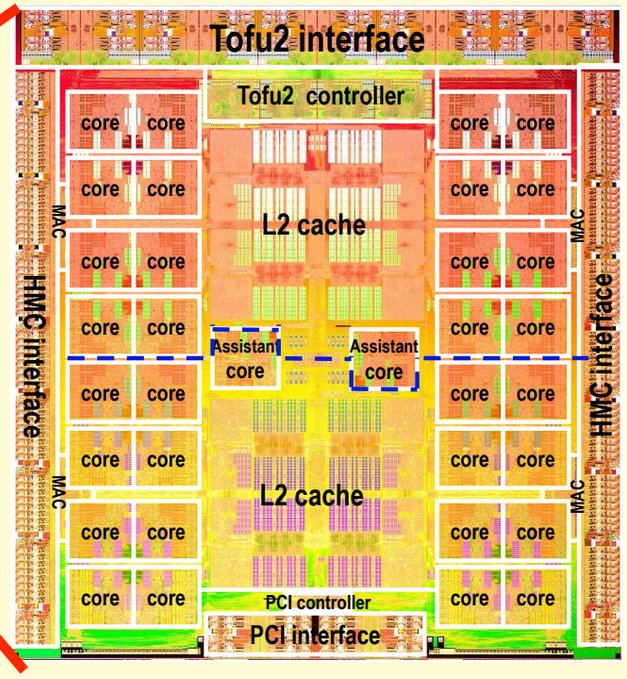


18 chassis (6,912 cores)

3 CPUs (Nodes)



4 system boards (384 cores)

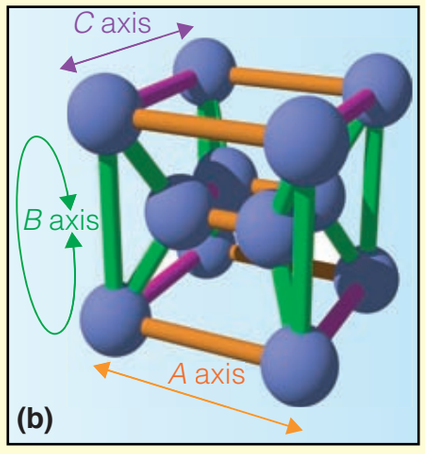


32+2 Cores + Tofu

Fujitsu FX100

- A chassis contains 3 nodes, switches and links.
- Tofu unit consists of 12 nodes is also a scheduling unit
  - Tofu 6D coordinate: “XYZabc” (a=2, b=3, c=2)
  - “XYZ” coord. represents the location of a Tofu unit
  - “abc” coord. represents the location inside of a Tofu unit

A Tofu unit



- A chassis contains 12 nodes
- 3 chassis compose 3 Tofu units

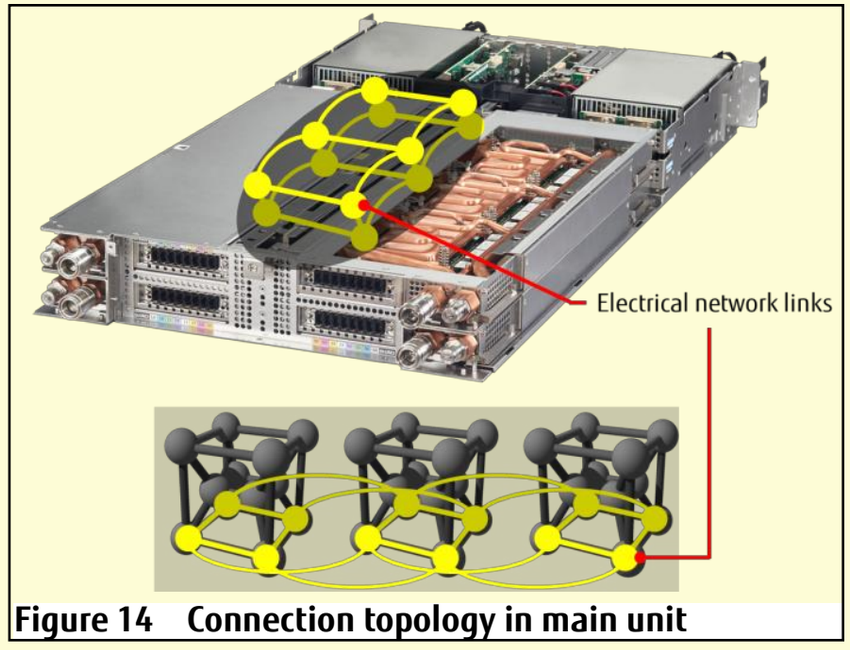


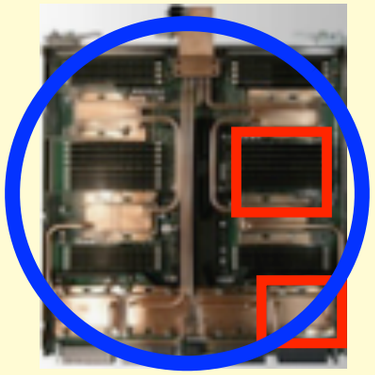
Figure 14 Connection topology in main unit

White paper FUJITSU Supercomputer PRIMEHPC FX100 – Evolution to the Next Generation
https://www.fujitsu.com/global/Images/primehpc-fx100-hard-en.pdf

The K and FX100

The K Computer 2011

24 chassis
(768 cores)

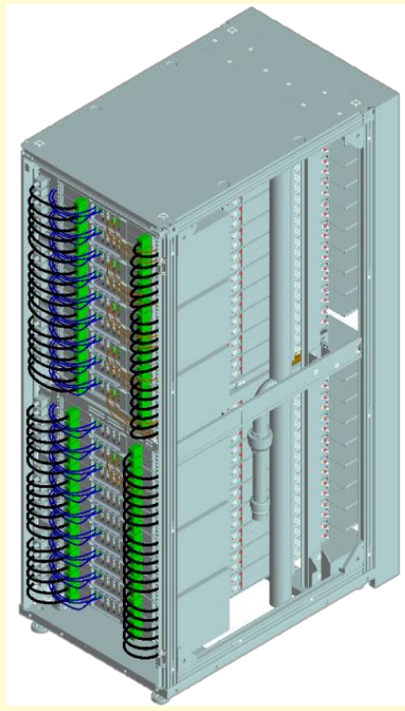


1 system board
4 CPUs and 4 ICCs
(32 cores)

The Tofu circuit is on the same CPU die, however, the Tofu circuit can keep running while the CPU cores are shutdown and power off.

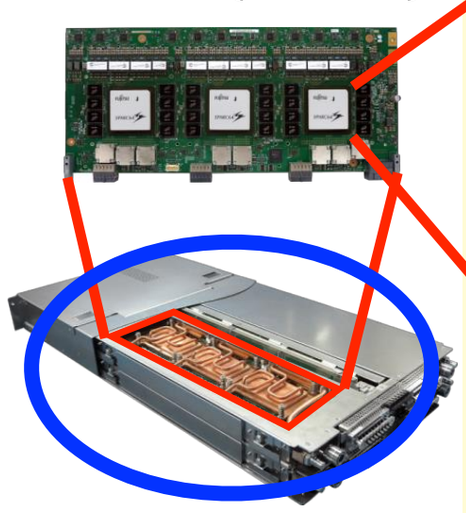
FX100 2015

http://accc.riken.jp/wp-content/uploads/2015/06/chiba.pdf

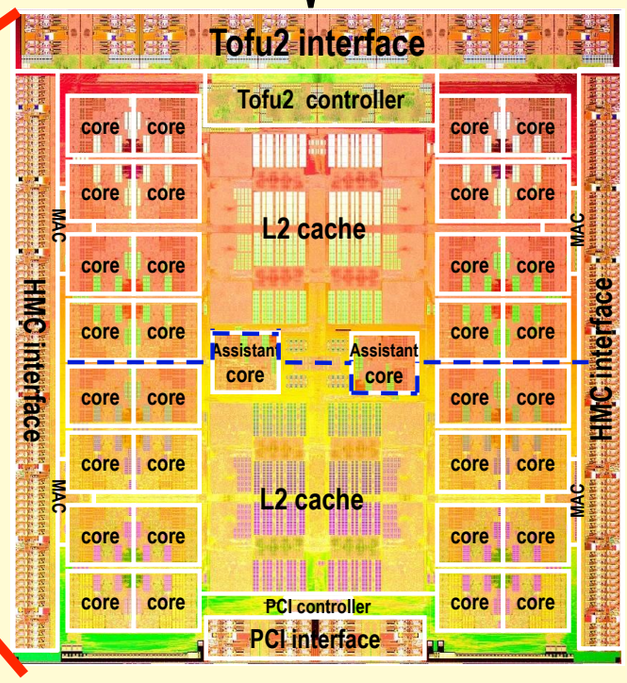


18 chassis (6,912 cores)

3 CPUs (Nodes)



4 system boards
(384 cores)



32+2 Cores + Tofu

Various Units in FX100

- Various units in Fujitsu FX100
  - Unit of network
    - Tofu (12 nodes)
  - Unit of scheduling
    - Tofu (not to interfere with other jobs)
  - Physical Unit
    - Chassis - A chassis spans 3 Tofu units
- Replacement of a chassis
  - Affects 3 Tofu units (36 nodes, 1152 cores)
    - Before replacement, the jobs running on the 36 nodes must be aborted
    - While in the replacement, the affected 36 nodes can not accept jobs
  - Tofu is ***direct network***, replacement can affect entire network because XYZ connections for I/O are lost
  - This $36\text{-}N_f$ nodes are called apparent failure (in this talk)

Repair Schedule

- Entire system must go on as much as possible
- Replacement may cause more *apparent* failures as packaging density increases
- Replacement cannot take place as soon as failure happens
  - Remedy for apparent failures is getting harder
  - The more frequent system service, the higher running cost
  - So, repair is scheduled once in a day, 2-3 times in a week, once in a week, and so on

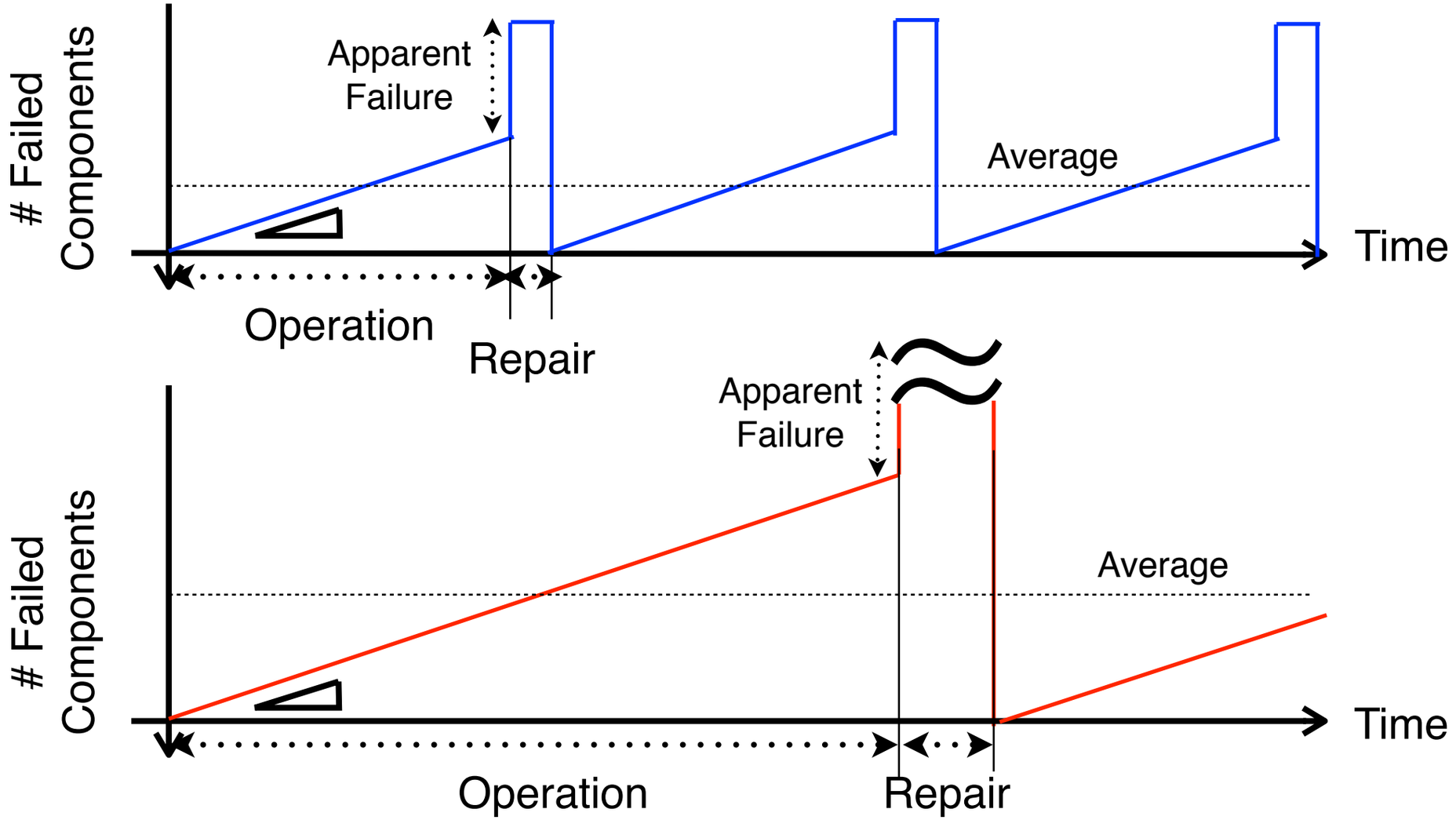
The K's case: Every morning, SEs replace the failed nodes

1. Shutdown the chassis
2. Unplug the chassis
3. Replace failed mother board
4. Plug the chassis
5. Reboot (K's nodes are disk-less)

Apparent failure

Repair Interval

- The longer repair interval, the larger number of failed parts
  - K: One node failure in 1~2 days



Towards Exa-flops

- Higher failure rate
  - larger number of components
    - end of Moore's law is close

- Longer time between repair
  - to reduce running cost
  - denser packaging results in more apparent failures
    - larger impact on running jobs

➡ Always having one or more number of failed components

Network Resilience Towards Exa-scale and Beyond

My Personal Opinion

Failure will be Daily Life

- Assumption of current HPC failure happens unexpectedly and unusually
  - System design is based on particular rules and algorithms
  - Random failure breaks those rules and algorithms
  - Node MTBF is less than a day already

- If failure is daily happening, why don't we design HPC systems having failures in mind ?

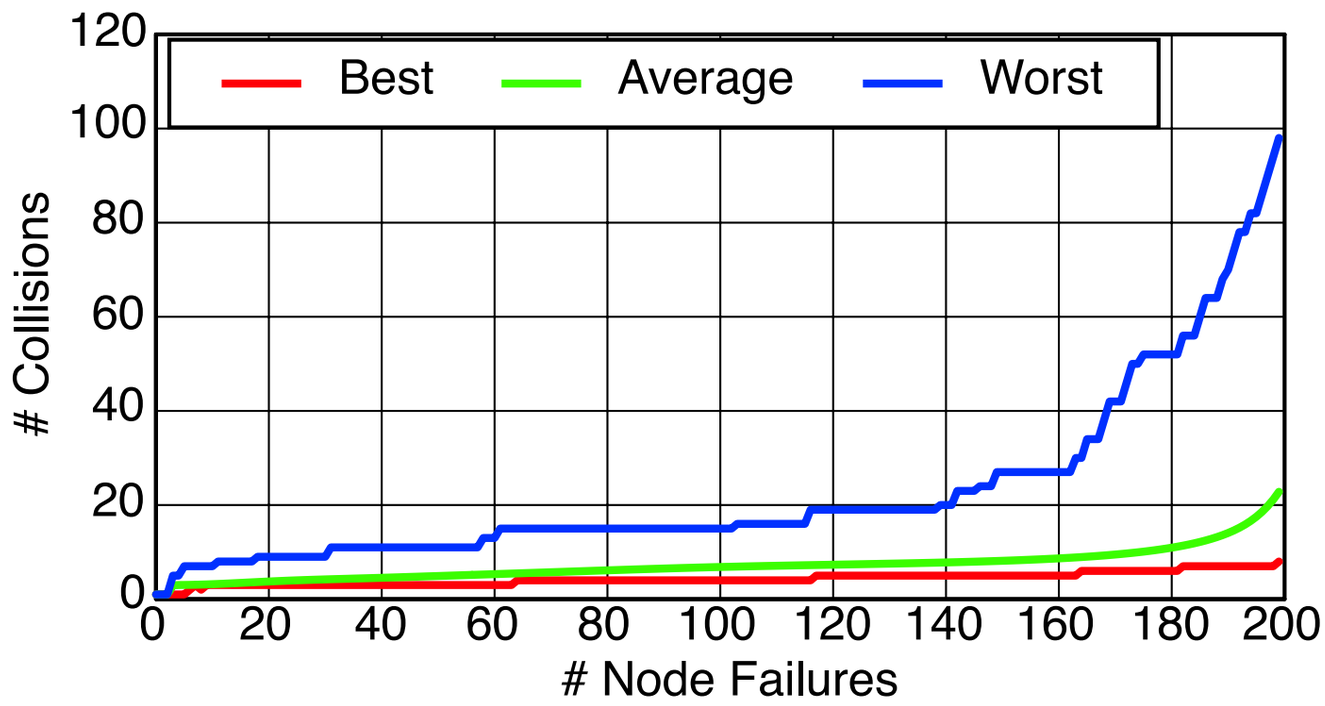
Failure Conscious Design

- Failure
  - happens randomly
  - # Combinations are *factorial !*

  - **impossible to handle failures case by case**
  - **impossible to predict performance degradation due to the failures**

Stencil and Cartesian Topology

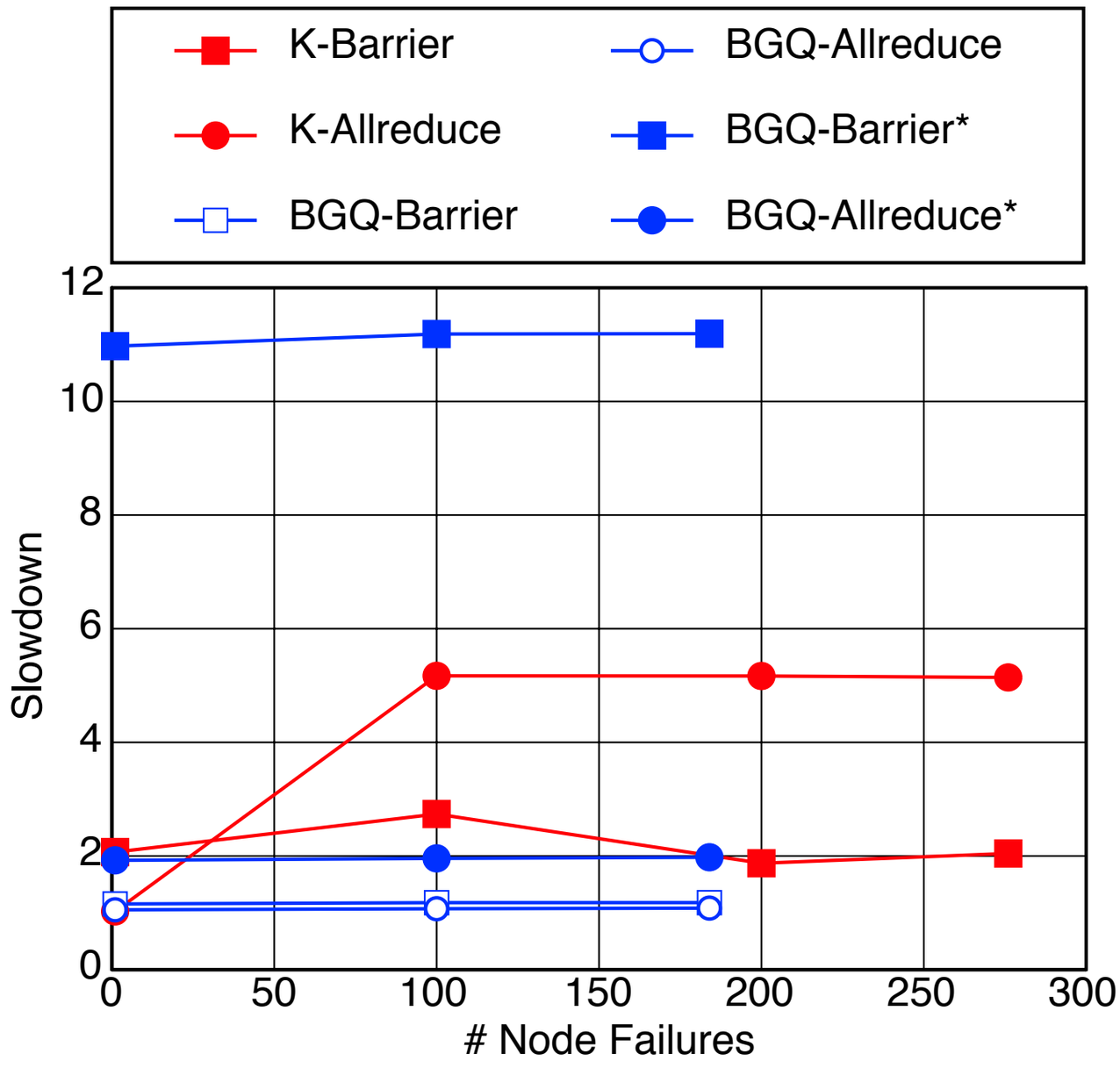
- The node failure problem in stencil computations is revisited
  - Communication pattern of stencil computation fits with Cartesian topology very very well
  - When spare node substitutions take place, then the fitness is gone and performance degrades



5P-Stencil Communication Performance Degradation over the Number of Failed Nodes [7]

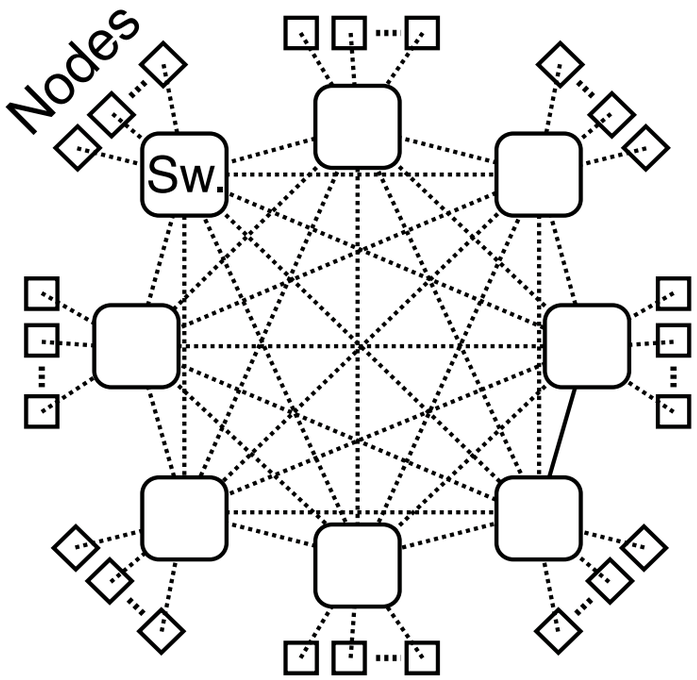
Topology and Protocol

- Protocols of collective operations are optimized according to topology
- If conditions of H/W support are NOT met, then general protocol takes place
- Failure break those conditions

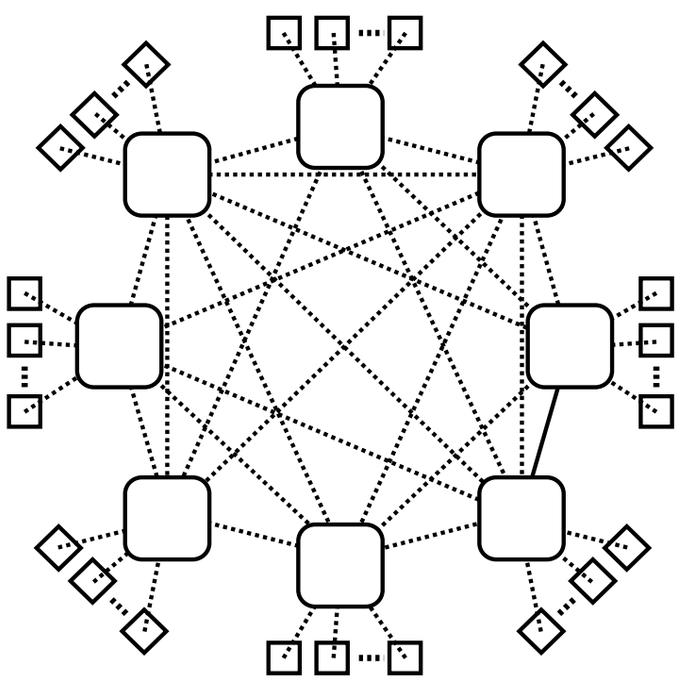


Regular topology turns into random topology

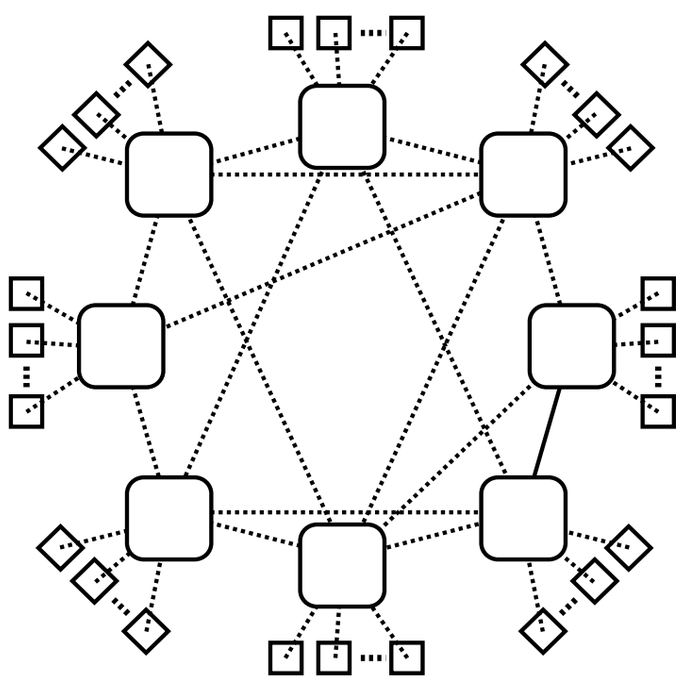
as the number of failed links increases



(Full) Dragonfly

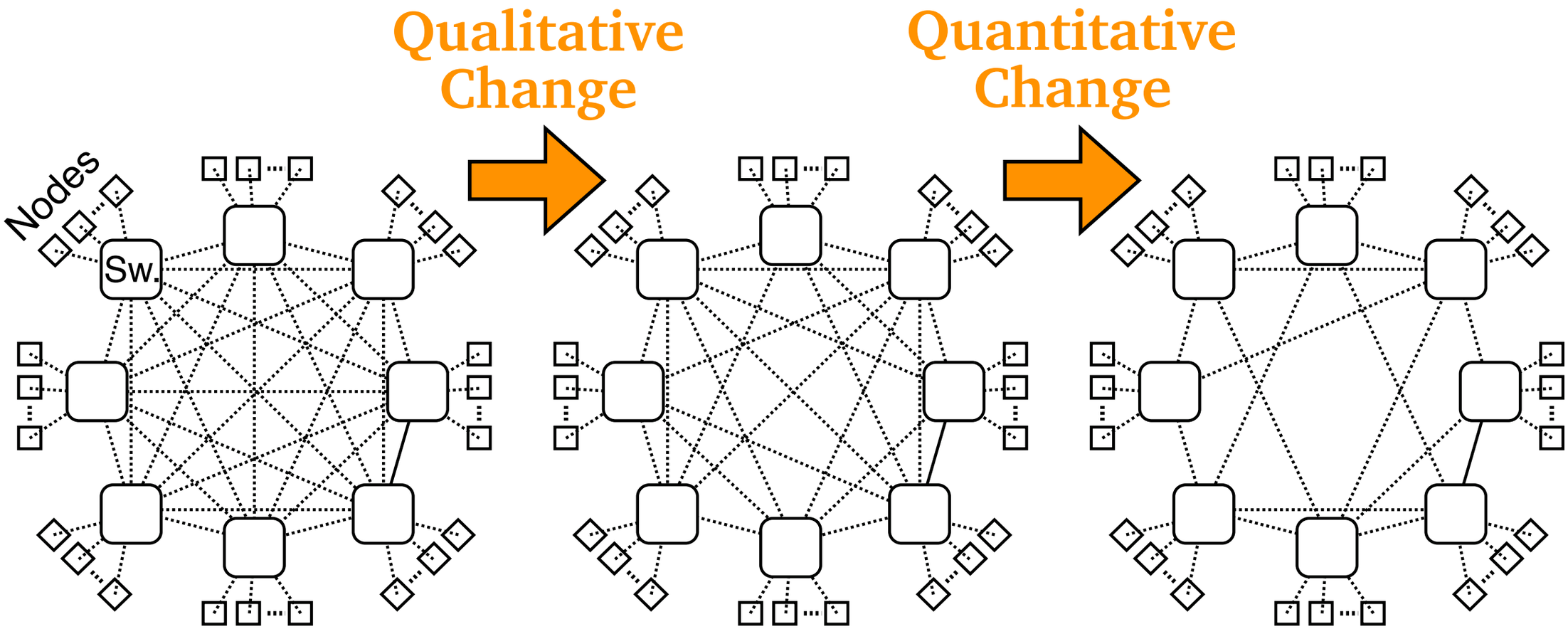


22/28



16/28

Regular topology turns into random topology as the number of failed links increases



(Full) Dragonfly **22/28** **16/28**

Randomness may be an answer

Randomness may be an answer

- Can we rely on the rules and algorithms which can be broken by failures ?
  - Failures on regularity

    Qualitative change: *Hard to imagine*

Randomness may be an answer

- Can we rely on the rules and algorithms which can be broken by failures ?
  - Failures on regularity

    Qualitative change: *Hard to imagine*
- What if we give up having such rules ?
  - Failures on randomness

    Quantitative change: *Easier to imagine*

Randomness may be an answer

- Can we rely on the rules and algorithms which can be broken by failures ?
  - Failures on regularity

    Qualitative change: *Hard to imagine*
- What if we give up having such rules ?
  - Failures on randomness

    Quantitative change: *Easier to imagine*
- *Let's start designing random systems from the beginning, forget about failures in regular systems*

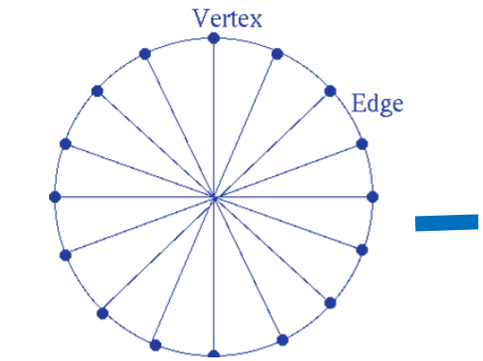
➡Random Topology

➡Random Network

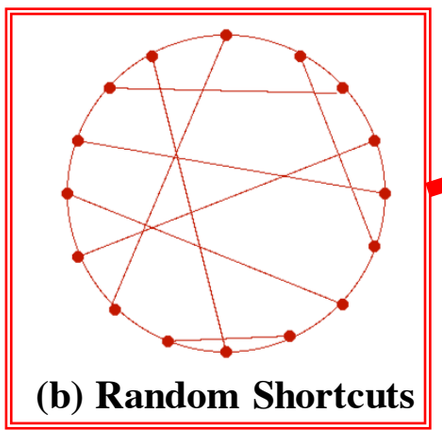
Random Topology (1)

Good Point of Random Topology

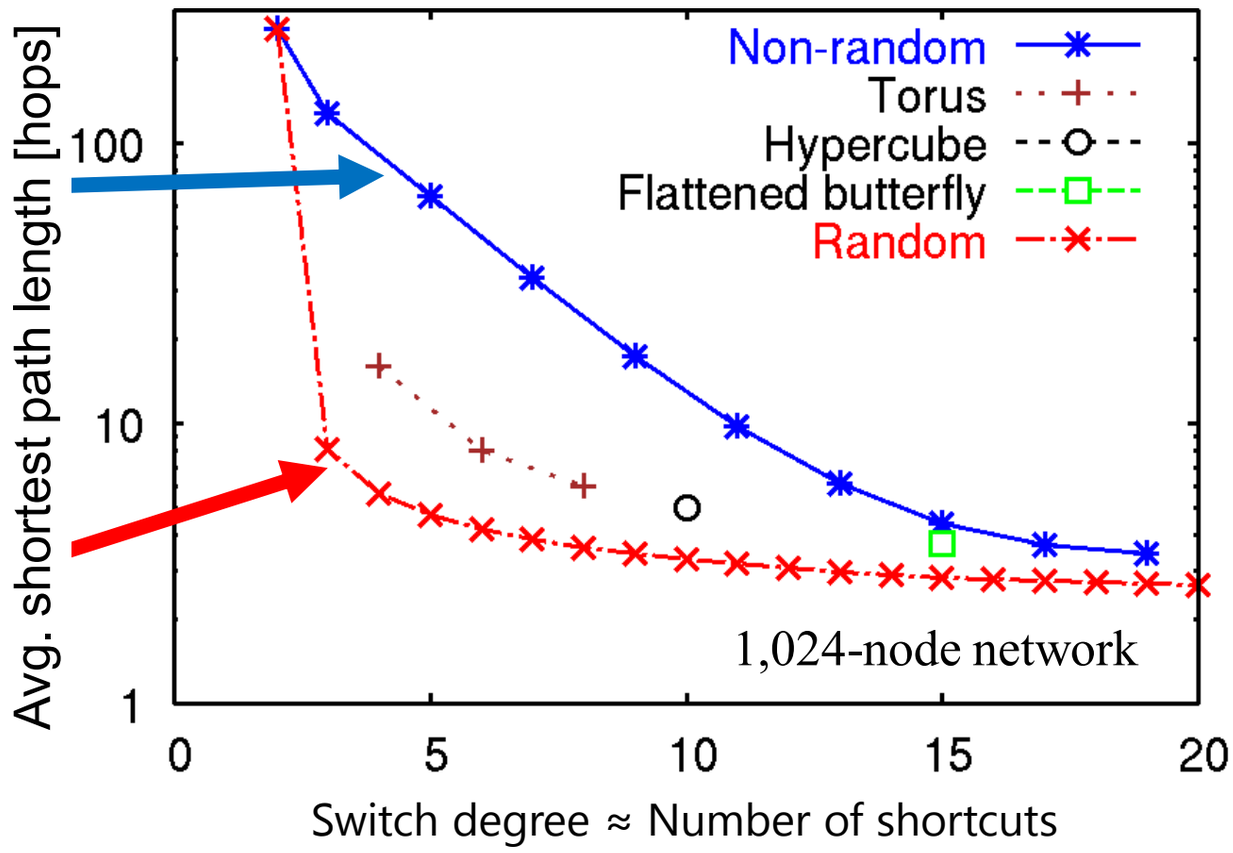
- Goal: to make a low-latency topology for HPC networks
  - low diameter and low average path hops
- Random topology is best [Koibuchi et al, ISCA2012]



(a) Non-random Shortcuts



(b) Random Shortcuts



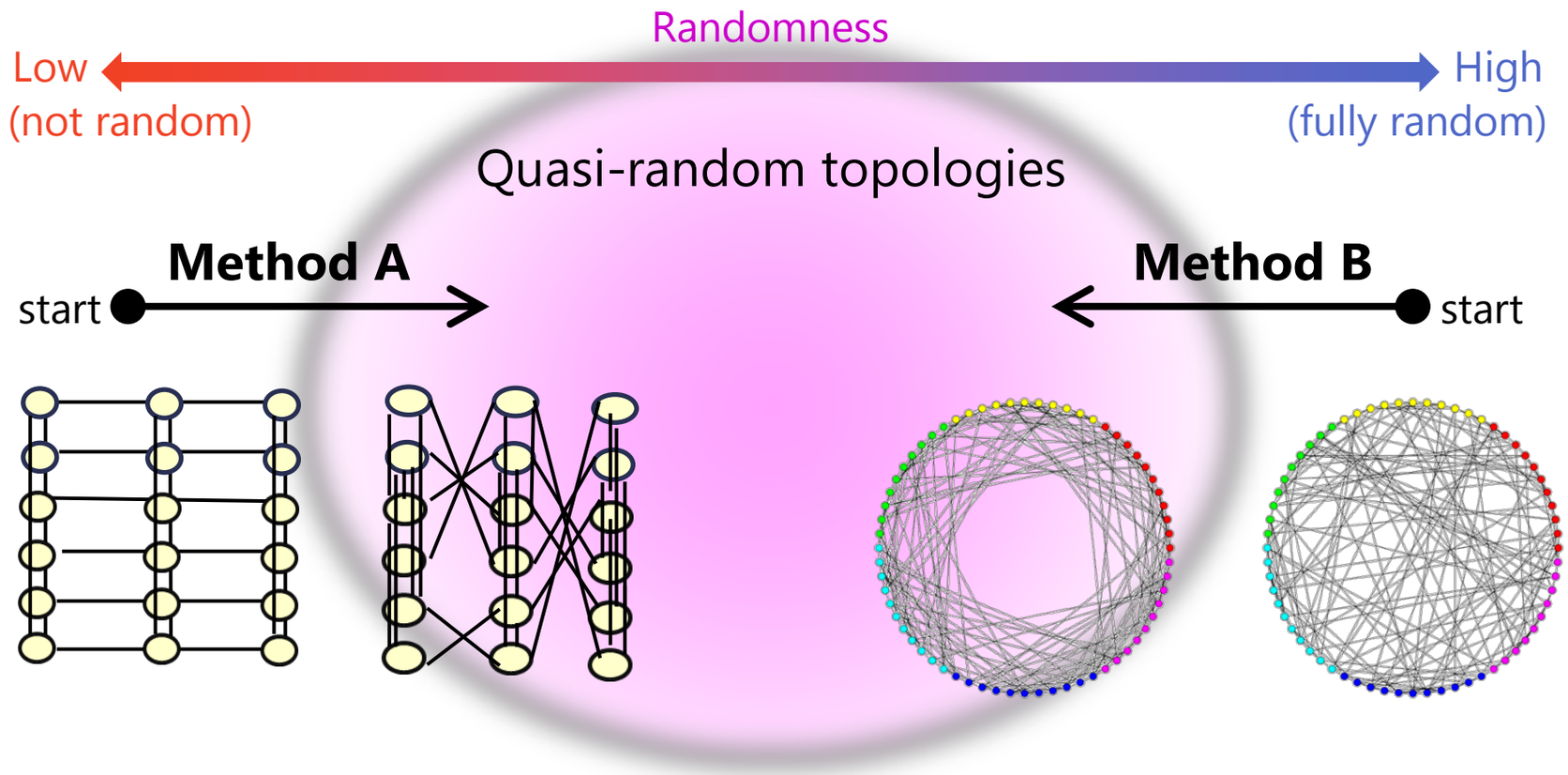
3

Michihiro Koibuchi, http://research.nii.ac.jp/~koibuchi/pdf/hpca2013_slide.pdf

Random Topology (2)

Two Approaches to Quasi-randomness

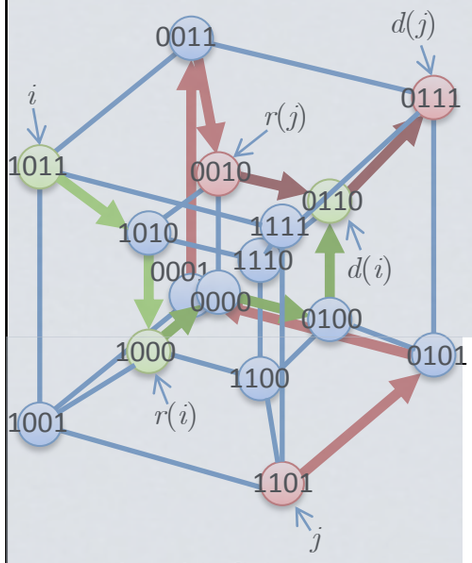
- **Method A** makes a non-random topology random
- **Method B** makes a random topology layout-friendly



Michihiro Koibuchi, http://research.nii.ac.jp/~koibuchi/pdf/hpca2013_slide.pdf

Random Routing in Hypercube

Random Routing in Hypercube



Random bit-fixing routing

i	$r(i)$	$d(i)$
0000	*0000*	0000
0001	*0001*	0100
0010	*1000*	1000
0011	*0101*	1100
0100	*0001*	0001
0101	*1110*	0101
0111	*1101*	1101
1110	*0000*	1011
1111	*1110*	1111

A two-stage configuration

- For deterministic bit-fix routing, the worst case requires at least $2^{n/2}/2$ steps (exponential in n)
- But for random bit-fix routing, it requires $\mathrm{O}(n)$ steps with high probability (i.e., using more than $\mathrm{O}(n)$ steps has a vanishing probability converging to 0, as $n \rightarrow 0$)
- Random bit-fix routing has two stages:
  1. Pick a random node $r(i)$ in the hypercube independently, and use bit-fixing routing from i to $r(i)$
  2. Use bit-fixing routing from $r(i)$ to $d(i)$
- Obviously, longer paths are needed for random bit-fix routing. Then why is this better?
- The intuition is that random routing can *average out* the worst case configuration from deterministic routing
- The probability that a randomly generated configuration is the worst case is very low, and is vanishing for large n
- This intuition is behind many randomized algorithms

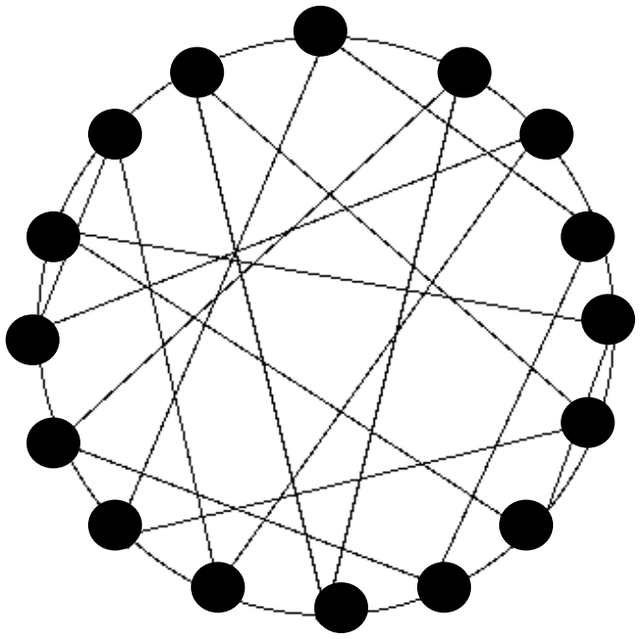
Sid C-K Chau, https://www.cl.cam.ac.uk/teaching/1011/CompSysMod/RandBits_Lec2V2.pdf

Dynamic routing vs. Random routing

- A switch has several routing candidates for a packet to go through
  - Static routing
    - choose fixed one always
  - Dynamic routing
    - choose one according to network status
- Random routing
  - choose one in a random way
    - not have to be uniformly random

Randomness in a Network

- Combination of regularity and randomness

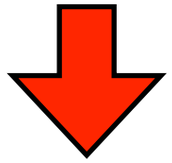


- Random Topology
  - Regular part + Random part
    - Ex) Ring + Random shortcuts
- Random (Oblivious) Routing
  (≠ Brownian motion)
  - Random routing + Regular routing
    - Node/switch on the way is randomly chosen

- Failure may happen on the regular part ?
  - The factorial nature can be relaxed
  - Ex) Redundant links of the regular part of topology.

My Last Word

"An eye for an eye, a tooth for a tooth"



Randomness for randomness

Randomness **MAY** save the future supercomputers *(not yet proven)*

Thank you

Reference

1) **High-radix router**: *Microarchitecture of a High-Radix Router*, John Kim, William J. Dally, *et. al.*, ISCA'05.
2) **Tofu network**: *THE TOFU INTERCONNECT*, Yuichiro Ajima, et. al., HOT INTERCONNECTS, 2012.
3) **Dragonfly network**: *Technology-Driven, Highly-Scalable Dragonfly Topology*, John Kim, William J. Dally, *et. al.,* ISCA '08.
4) **Routing algorithms**: *A Survey and Evaluation of Topology-Agnostic Deterministic Routing Algorithms*, J. Flich et al., in IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 3, pp. 405-425, March 2012.
5) **Shortest path finding algorithm (Dijkstra Algorithm)**: *A note on two problems in connexion with graphs*, Dijkstra, E.W., In Numerische Mathematik, 1959.
6) **Adaptive routing in Infiniband**: *Fail-in-place Network Design: Interaction Between Topology, Routing Algorithm and Failures*, J. Domke, T. Hoefler, and S. Matsuoka, SC '14, 2014.
7) **Spare node substitution**: *Sliding Substitution of Failed Nodes*, Atsushi Hori, et. al., In Proceedings of the 22nd European MPI Users' Group Meeting, ACM, 2015.
8) **Random algorithms including the random routing:** *Randomized Algorithms*, Rajeev Motwani and Prabhakar Raghavan, Cambridge University Press, 1995.
9) **Random network**: *A Case for Random Shortcut Topologies for HPC Interconnects*, Michihiro Koibuchi, *et. al.*, ISCA'12.
10) **Another view on HPC network robustness**: *Robustness Attributes of Interconnection Networks for Parallel Processing*, Behrooz Parhami, Keynote lecture, 1st Int'l Suprcomputing Conf. (ISUM-2010), 2010 March 4. (https://www.ece.ucsb.edu/~parhami/pres_folder/parh10-isum-robustness-int-nets.ppt)