USC Viterbi School of Engineering | Google

Condor

Better Topologies through Declarative Design

Brandon Schlinker[1,2]
Radhika Niranjan Mysore[1], Sean Smith[1], Jeffrey C. Mogul[1], Amin Vahdat[1], Minlan Yu[2], Ethan Katz-Bassett[2], and Michael Rubin[1]

[1] Google Inc., [2] University of Southern California

SIGCOMM 2015

Our Focus:

Designing Topologies for **Large** Datacenters

Our Focus:

Designing Topologies for **Large** Datacenters

(10k - 100k+ servers)

Designing Topologies for **Large** Datacenters

Why exploring the topology design space is *hard*

How today's ad-hoc approach leads to suboptimal deployments

How Condor's systematic approach helps architects

Enabling efficient exploration and evaluation of the design space

Designing Topologies for **Large** Datacenters

Why exploring the topology design space is *hard*

How today's ad-hoc approach leads to suboptimal deployments

How Condor's systematic approach helps architects

Enabling efficient exploration and evaluation of the design space

"We need a datacenter for 50k servers"

Network Architect

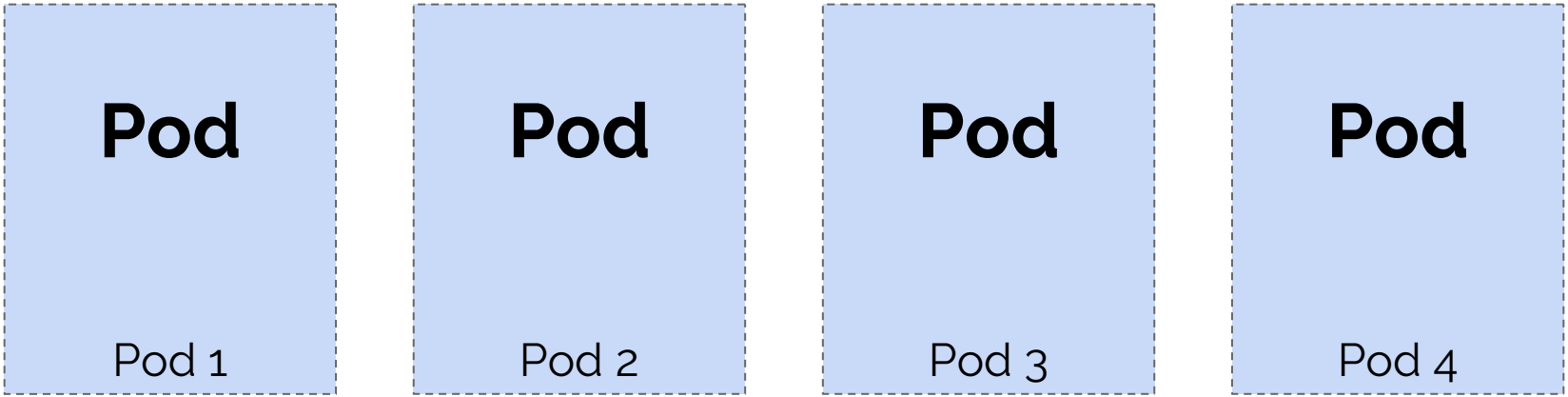
"We need a datacenter for 50k servers"

Network Architect

Let's use a **Fat-tree** topology!

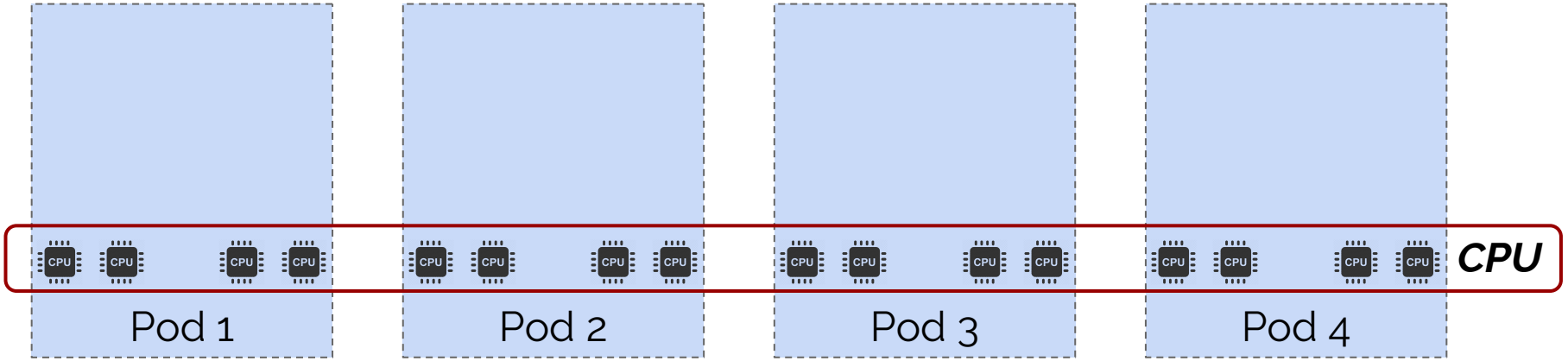
(SIGCOMM 2008)

The Fat-tree Topology



Each fat-tree contains 2+ **pods**

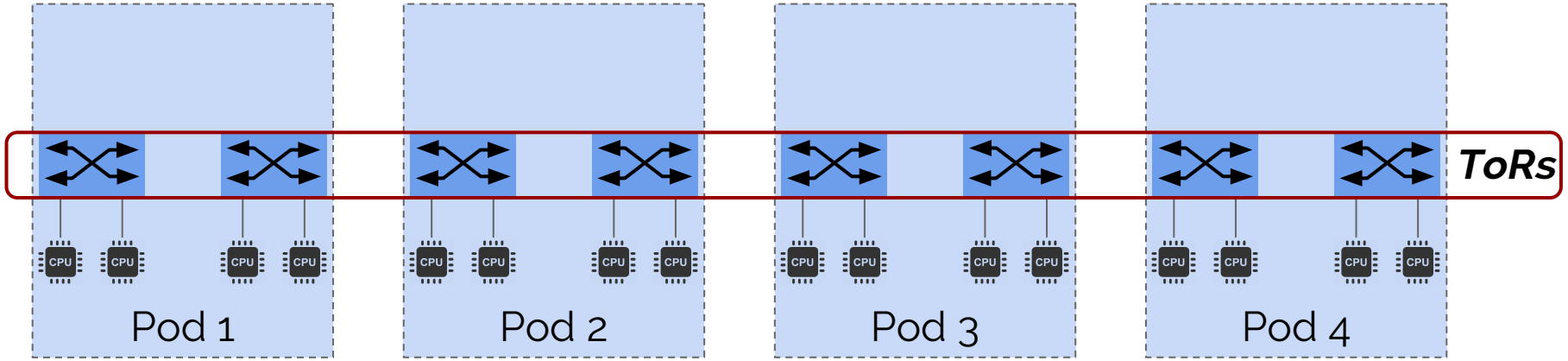
The Fat-tree Topology



Each **pod** contains:

Servers

The Fat-tree Topology

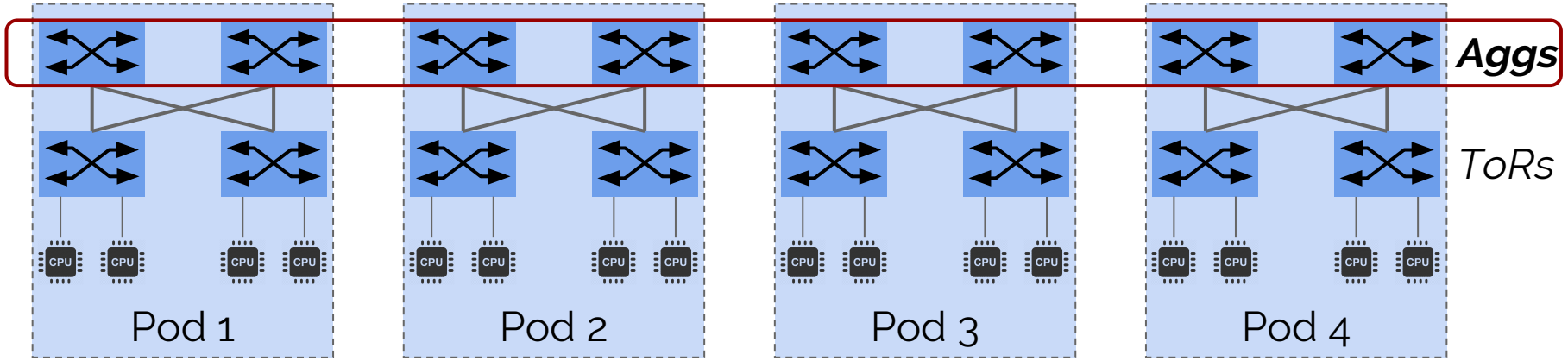


Each **pod** contains:

Servers

Top of Rack Switches

The Fat-tree Topology



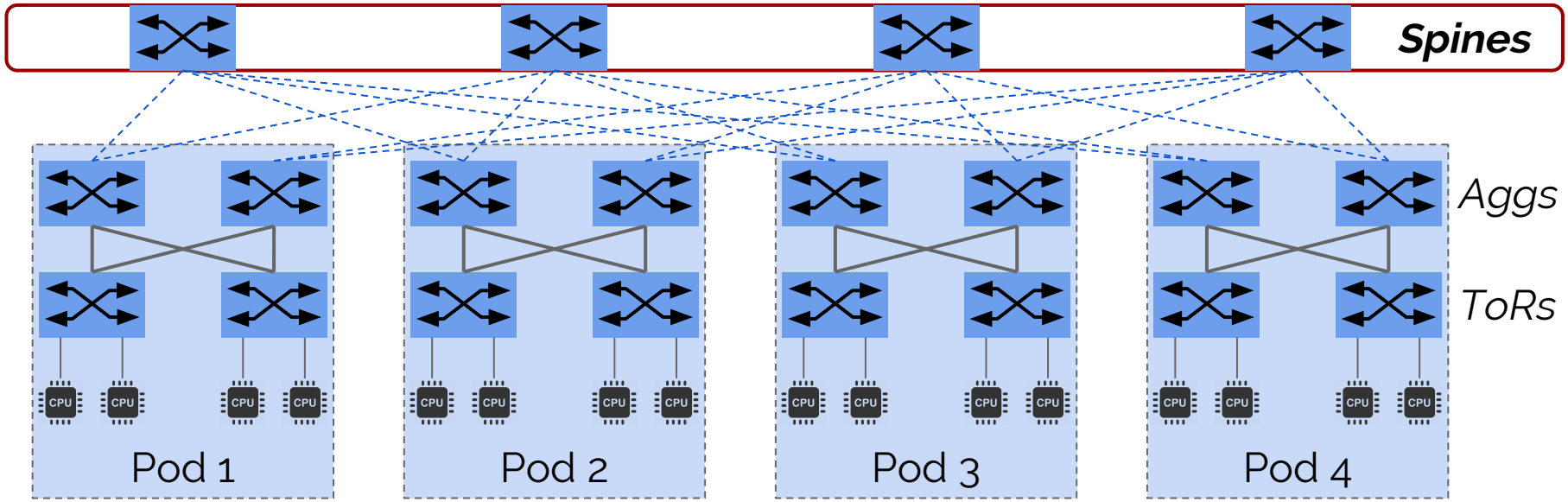
Each **pod** contains:

Servers

Top of Rack Switches

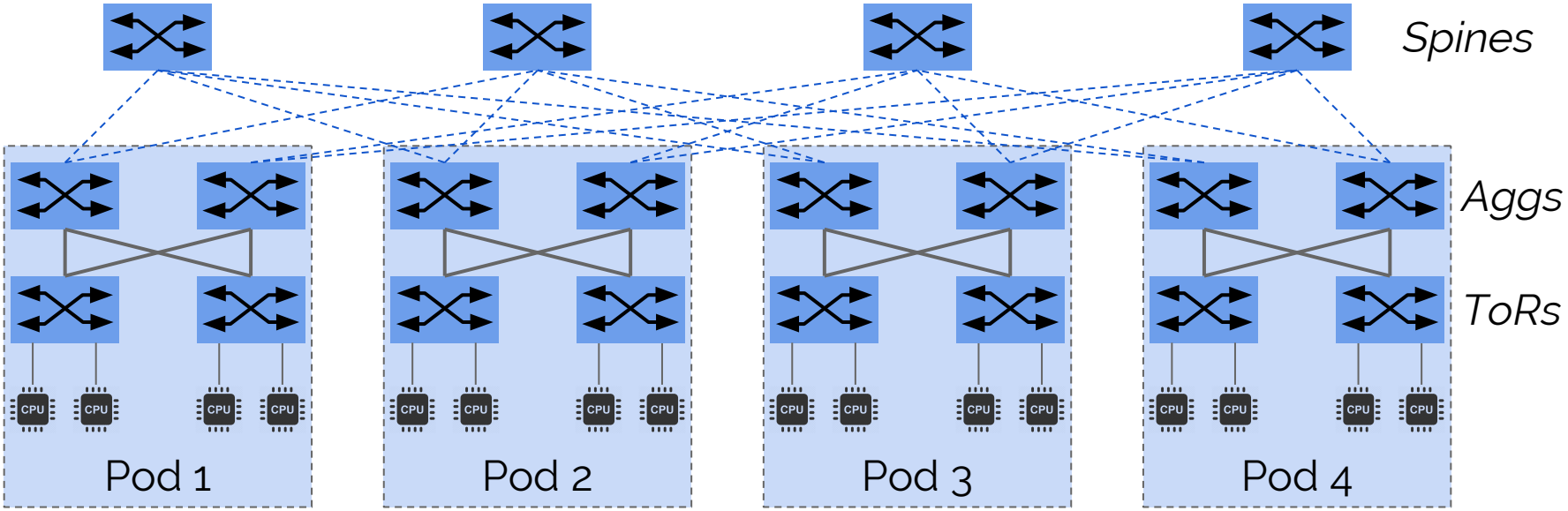
Aggregation Switches

The Fat-tree Topology



Pods are connected by **Spine Switches**

The Fat-tree Topology



Done.

The topology is deployed.

Up and Running...

Months go by...

The network runs fine most of the time...

Up and Running...

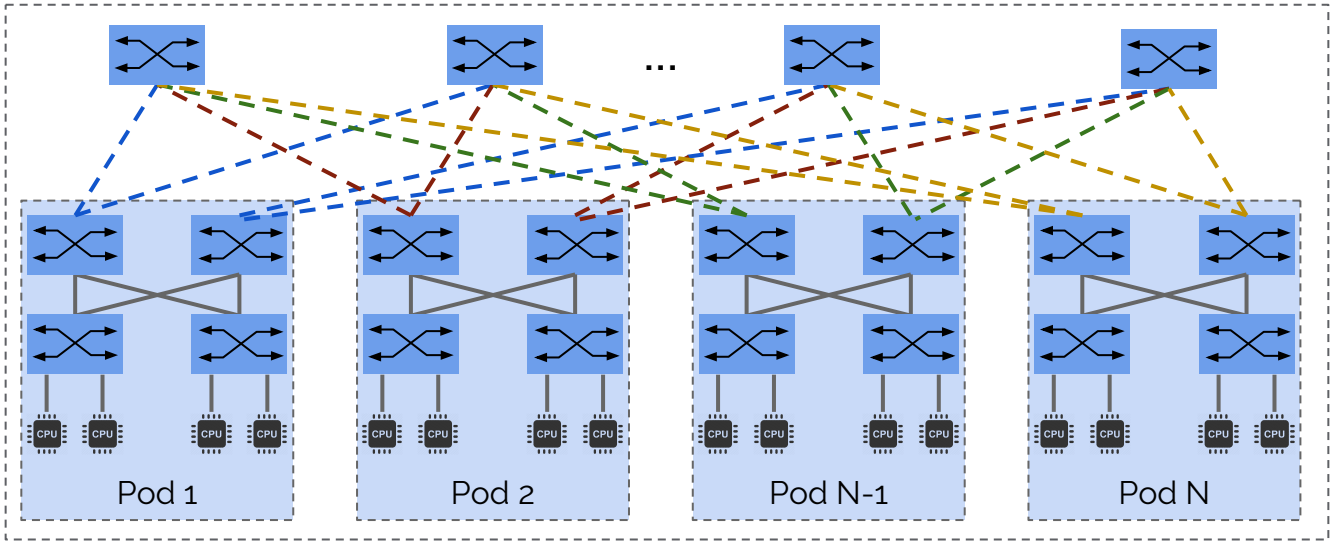
Months go by...

The network runs fine most of the time...

But when a failure happens

recovery takes longer than expected

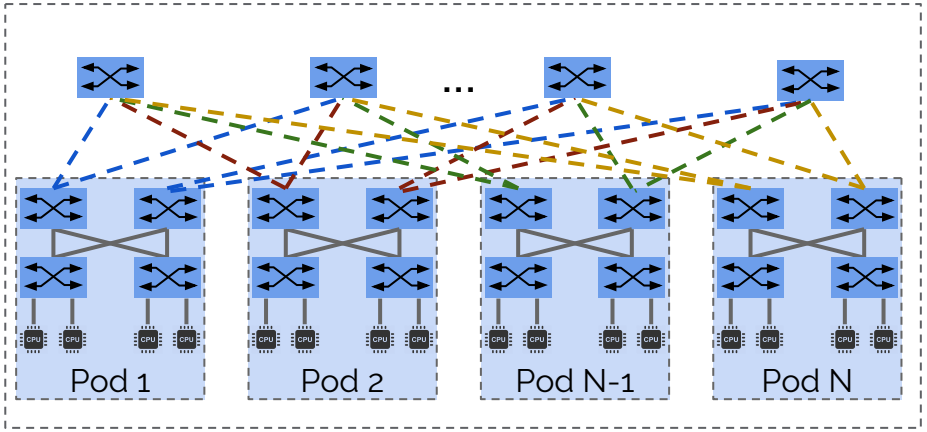
Hindsight is *20/20*



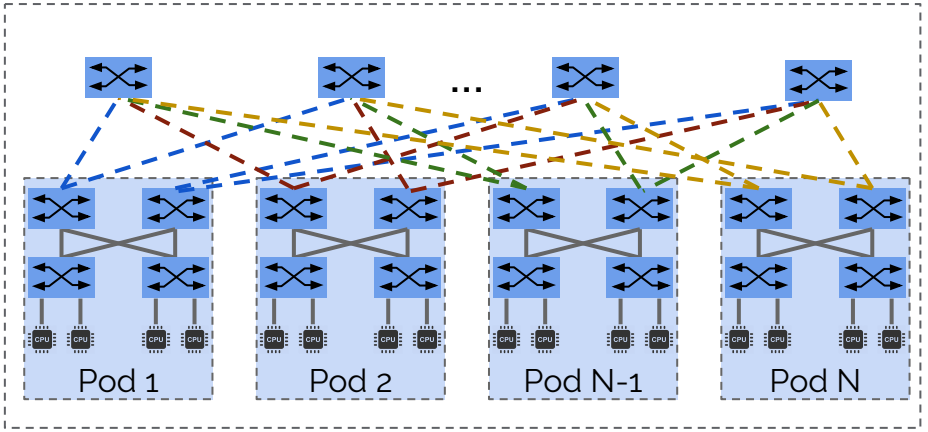
Fat-tree topology | SIGCOMM 2008

Architect goes back and reviews the design…

Hindsight is *20/20*

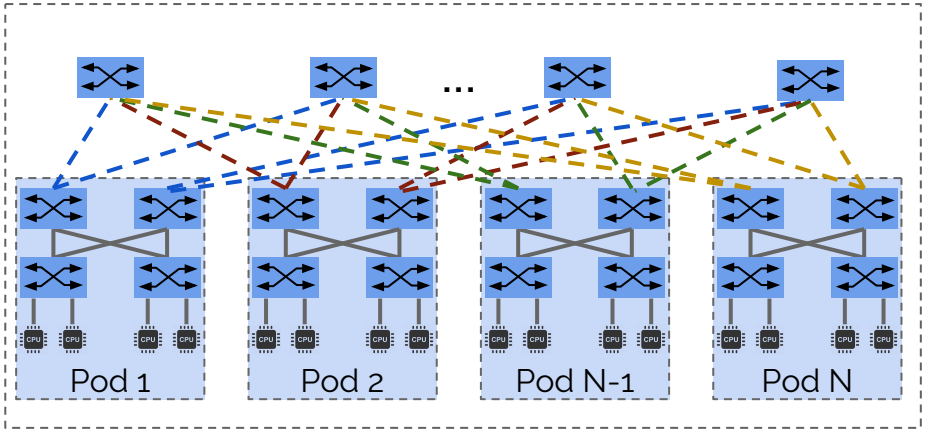


Fat-tree topology | SIGCOMM 2008

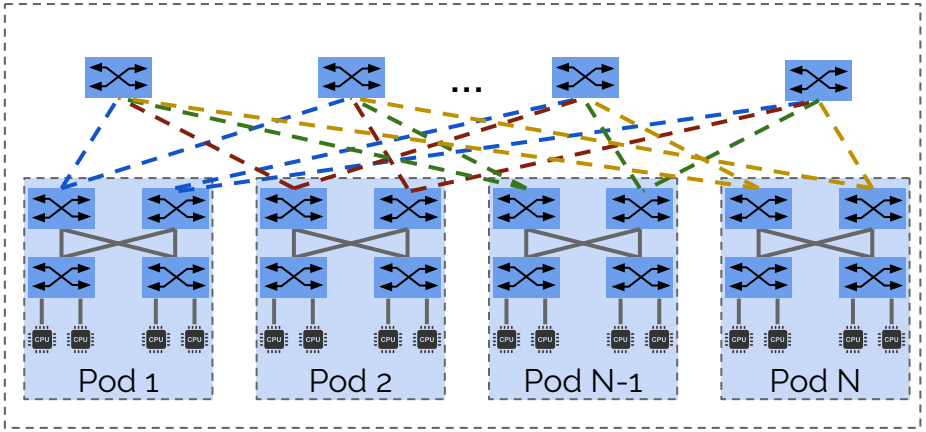


F10 topology | NSDI 2013

And discovers a ***small* wiring change** (*F10*)
that **allows faster recovery** from failures



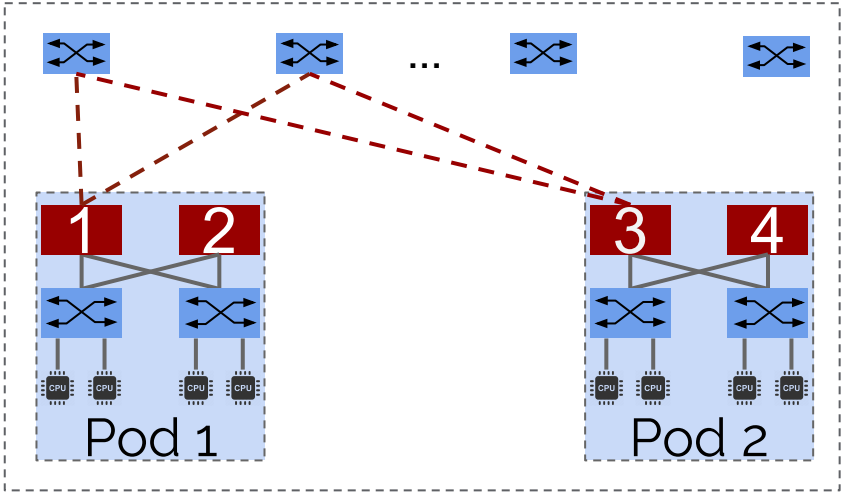
Fat-tree topology | SIGCOMM 2008



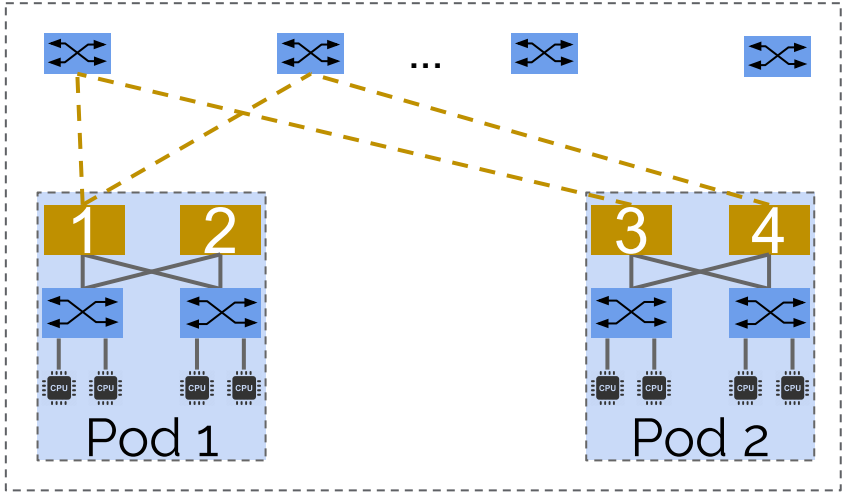
F10 topology | NSDI 2013

Why is F10's wiring better?

F10's *Hidden* Advantage



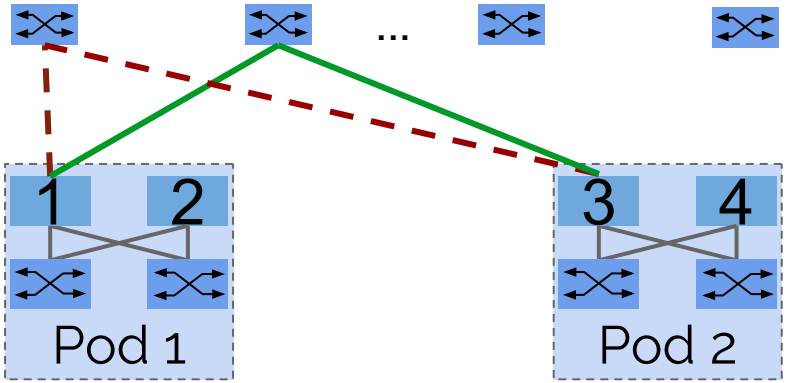
Fat-tree topology | SIGCOMM 2008



F10 topology | NSDI 2013

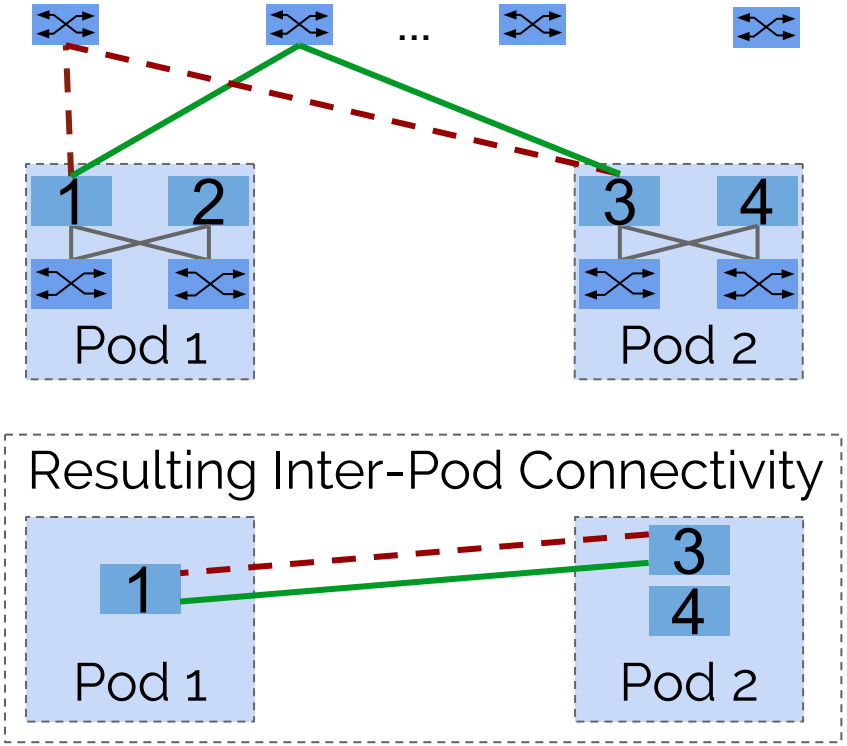
F10's small change improves
the connectivity between pods

F10's *Hidden* Advantage



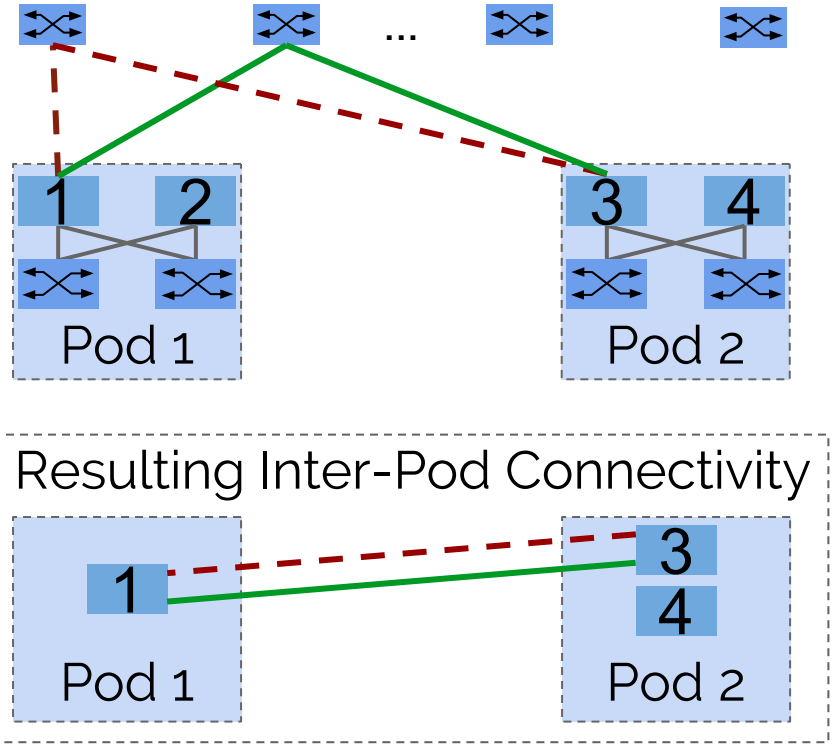
Fat-tree topology | SIGCOMM 2008

F10's *Hidden* Advantage

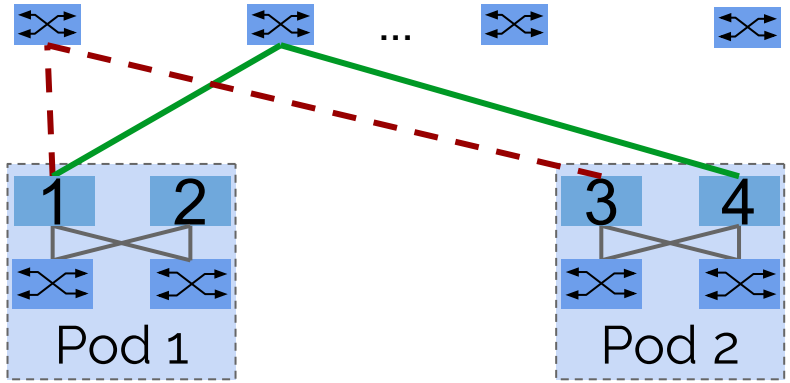


Fat-tree topology | SIGCOMM 2008

F10's *Hidden* Advantage

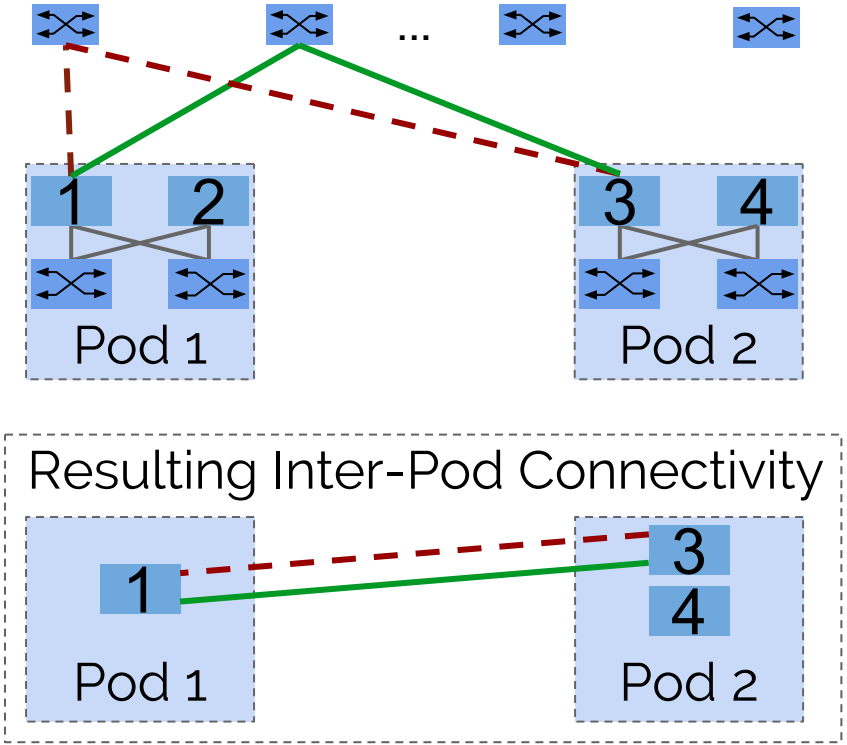


Fat-tree topology | SIGCOMM 2008

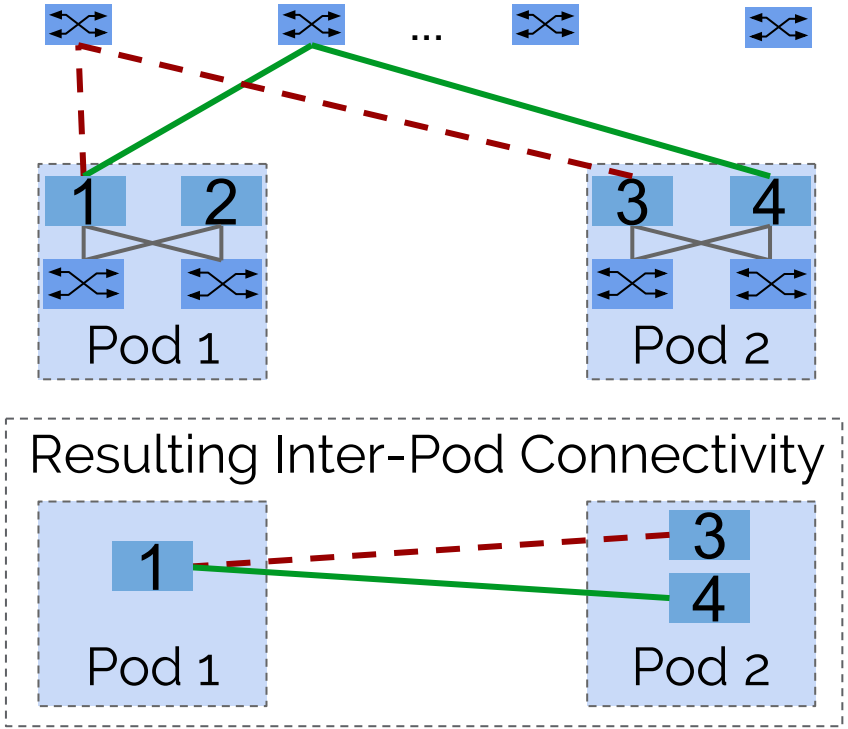


F10 topology | NSDI 2013

F10's *Hidden* Advantage

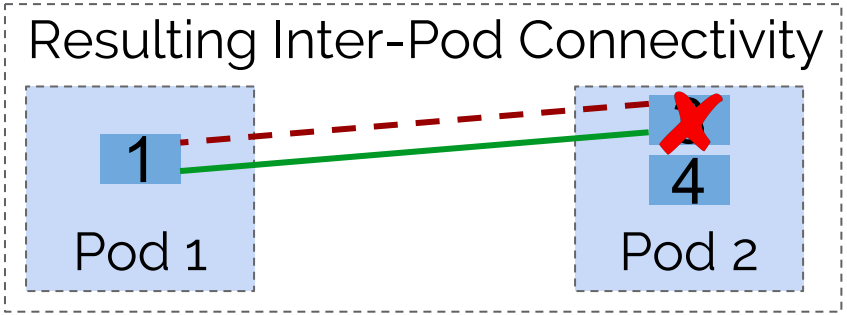


Fat-tree topology | SIGCOMM 2008

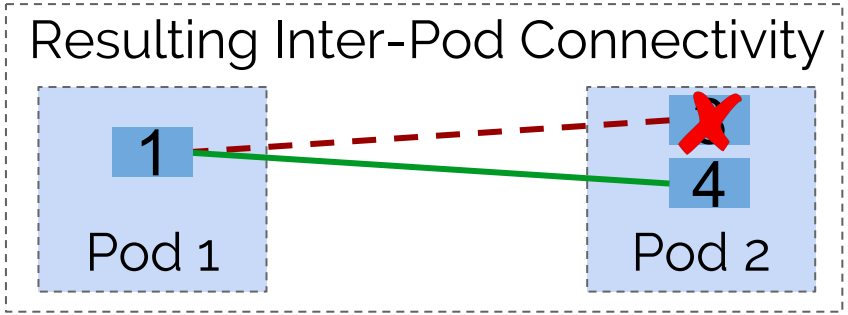


F10 topology | NSDI 2013

F10's *Hidden* Advantage



Standard wiring = **one path**
(via agg_3)



F10's wiring = **two paths**
(via agg_3 and agg_4)

F10's simple wiring change provides
greater path diversity, **faster recovery** from failure

Oops.


Oops.

These late realizations occur all *too often*

Why?

Architects make *lots* of decisions during design,
every decision can have **substantial impact**

Architects make *lots* of decisions during design,
every decision can have **substantial impact**

But it's **difficult to evaluate** if a **decision is *good***
so architects are **often blind to design defects**

One of architect's primary concerns is
Service Level Objective (SLO) Compliance

examples:
≥ 100 Mbps **throughput** 99.99% of the time
≤ 10 ms **latency** 99.99% of the time

One of architect's primary concerns is
Service Level Objective (SLO) Compliance

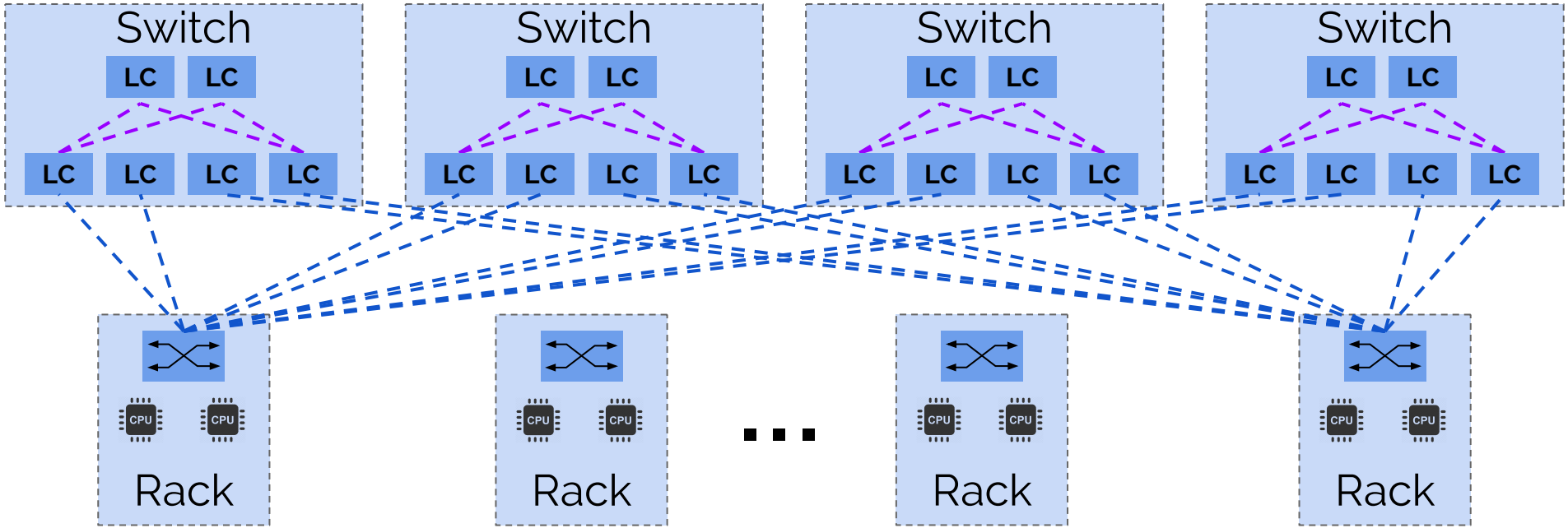
examples:
≥ 100 Mbps **throughput** 99.99% of the time
≤ 10 ms **latency** 99.99% of the time

Tough to estimate and compare SLO compliance

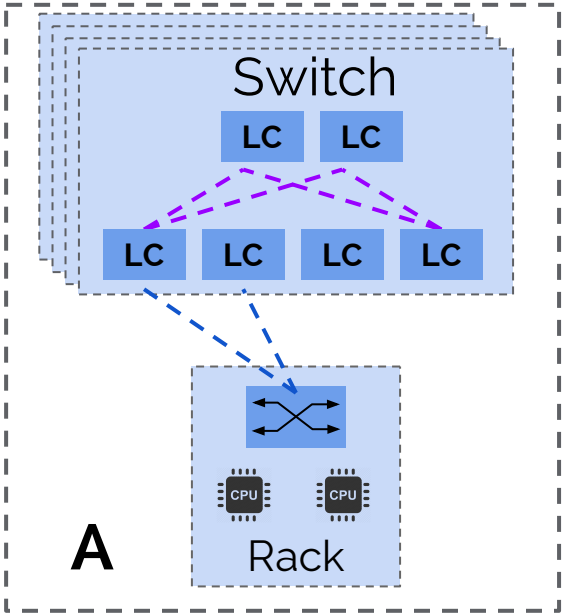
Challenge: Estimating SLO compliance of two designs

Example Topology:

Bunch of racks interconnected with big switches

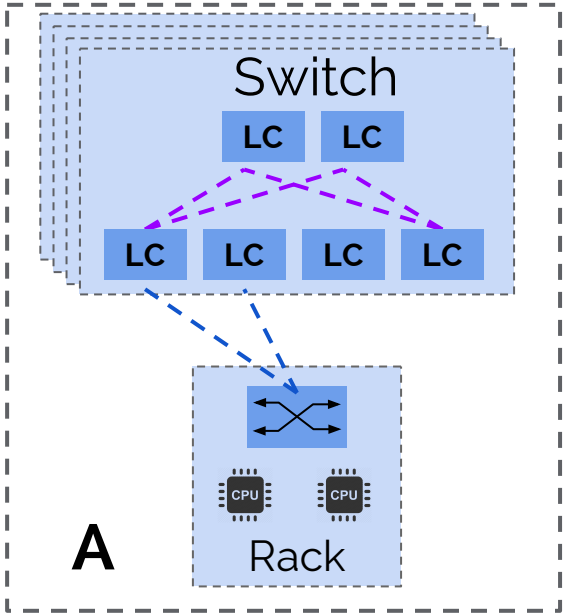


Challenge: Estimating SLO compliance of two designs

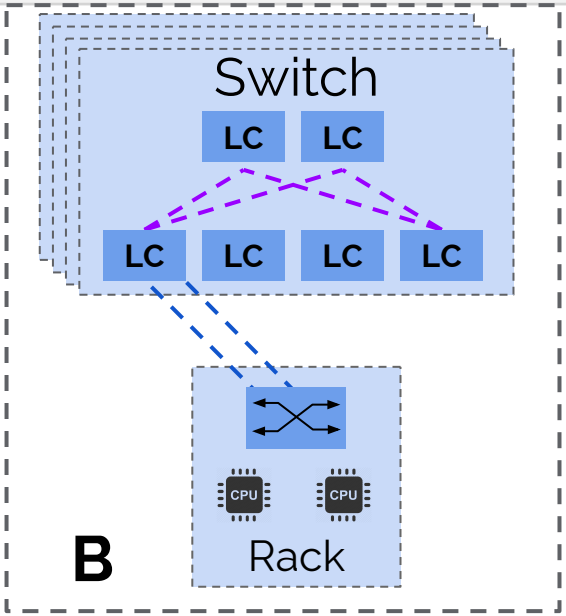


Option A:
Connect each ToR to ***two*** **linecards**
in each fabric switch

Challenge: Estimating SLO compliance of two designs

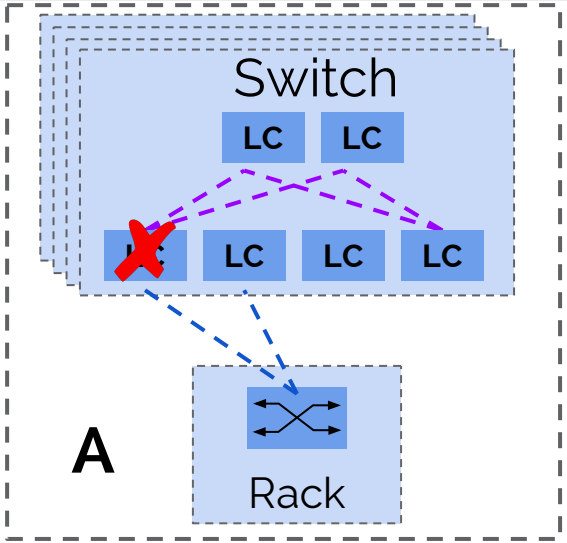


Option A:
Connect each ToR to ***two*** **linecards** in each fabric switch



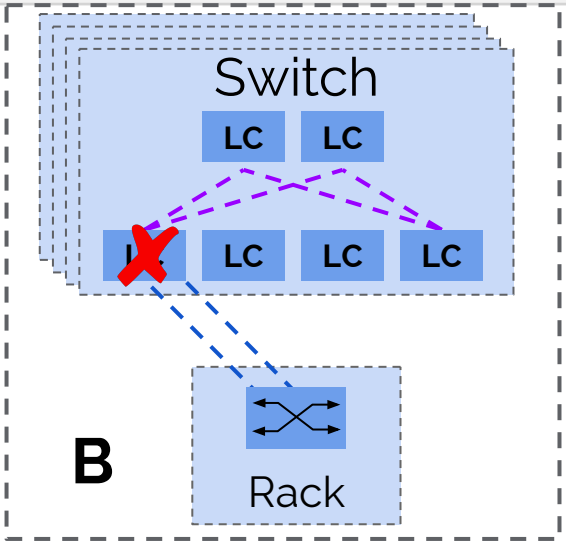
Option B:
Connect ToR to ***same*** **linecard twice** in each fabric switch

Challenge: Estimating SLO compliance of two designs



Linecard Failure

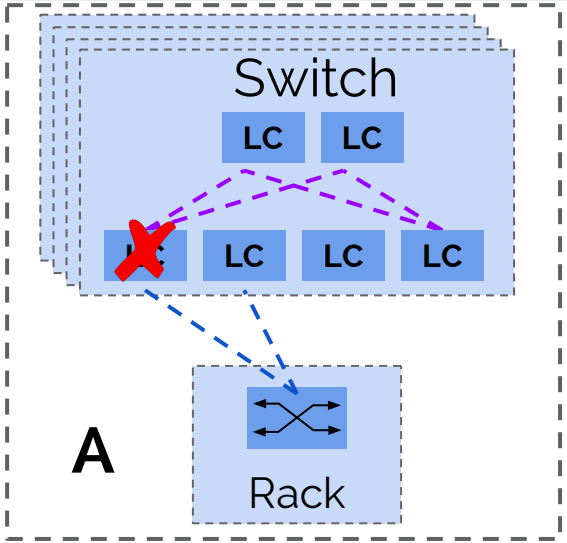
Rack loses **1/8th** of inter-rack throughput (1 out of 8 links lost)



Linecard Failure

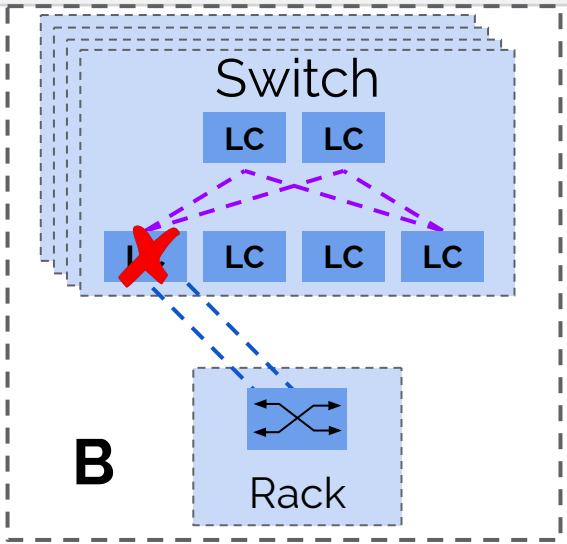
Rack loses **1/4th** of inter-rack throughput (2 out of 8 links lost)

Challenge: Estimating SLO compliance of two designs



Linecard Failure

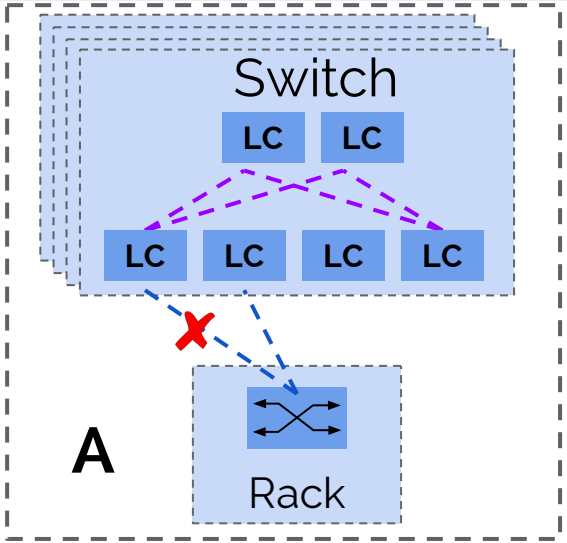
Rack loses 1/8th of inter-rack throughput (1 out of 8 links lost)



Linecard Failure

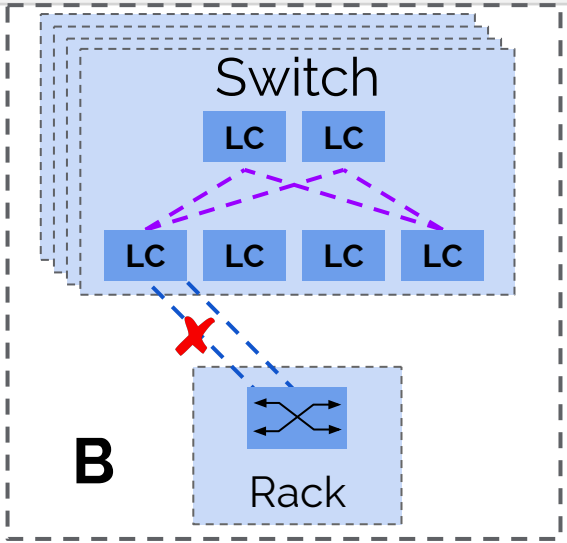
Rack loses **1/4th** of inter-rack throughput (2 out of 8 links lost)

Challenge: Estimating SLO compliance of two designs



Link Failure

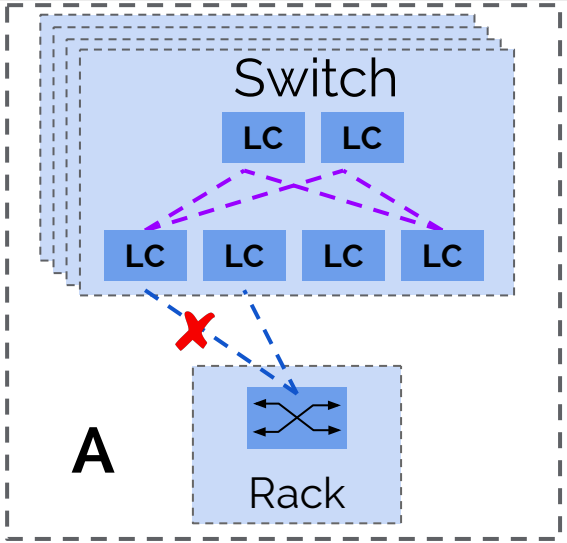
Routing reconvergence and packet loss



Link Failure

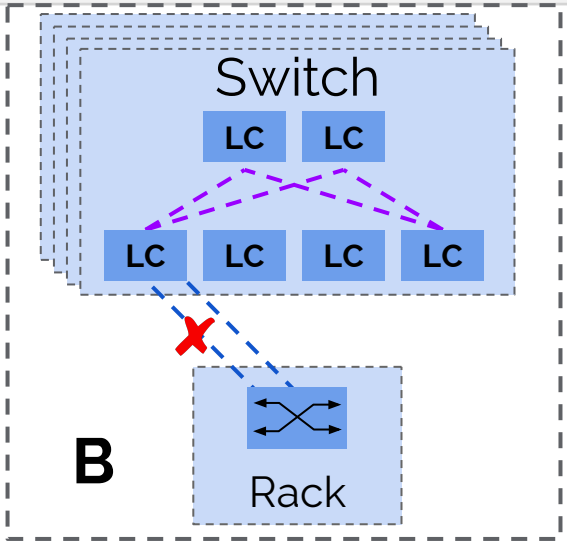
Local failure handling and no packet loss

Challenge: Estimating SLO compliance of two designs



Link Failure

Routing reconvergence and packet loss



Link Failure

Local failure handling and no packet loss

	Design A	Design B
Line Card Failure		✗
Link Failure	✗	

Is either design SLO compliant?
Which design is *better*?

	Design A	Design B
Line Card Failure		✗
Link Failure	✗	

Is either design SLO compliant?
Which design is *better*?

Depends on SLO, failure / recovery rates, routing protocols

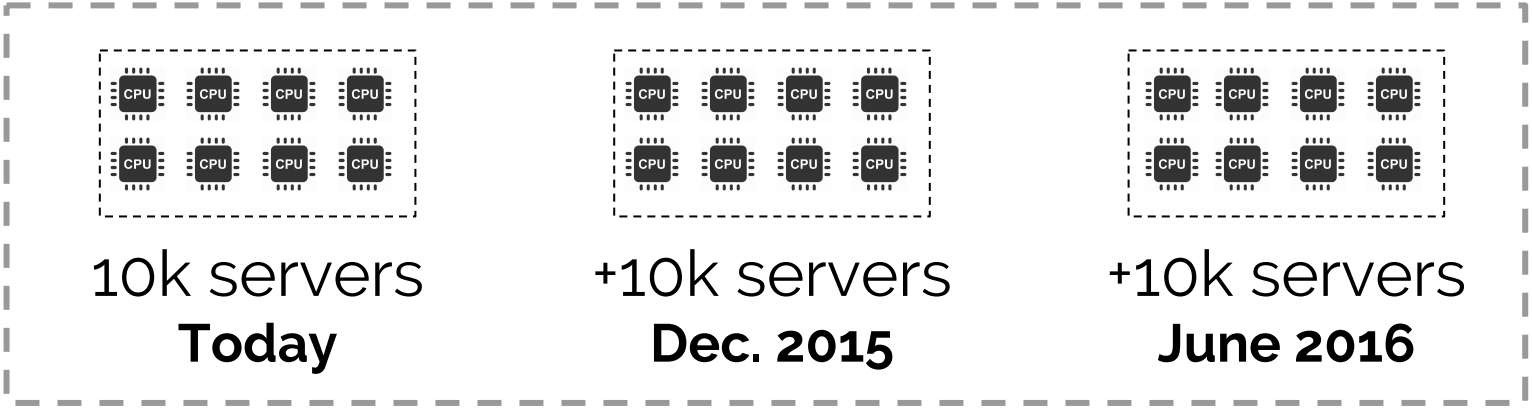
cannot evaluate via simple calculations

Depends on SLOs, failure and recovery rates, protocols

cannot evaluate via simple calculations

architects are often blind to design defects

Challenge: Designing Expandable Topologies



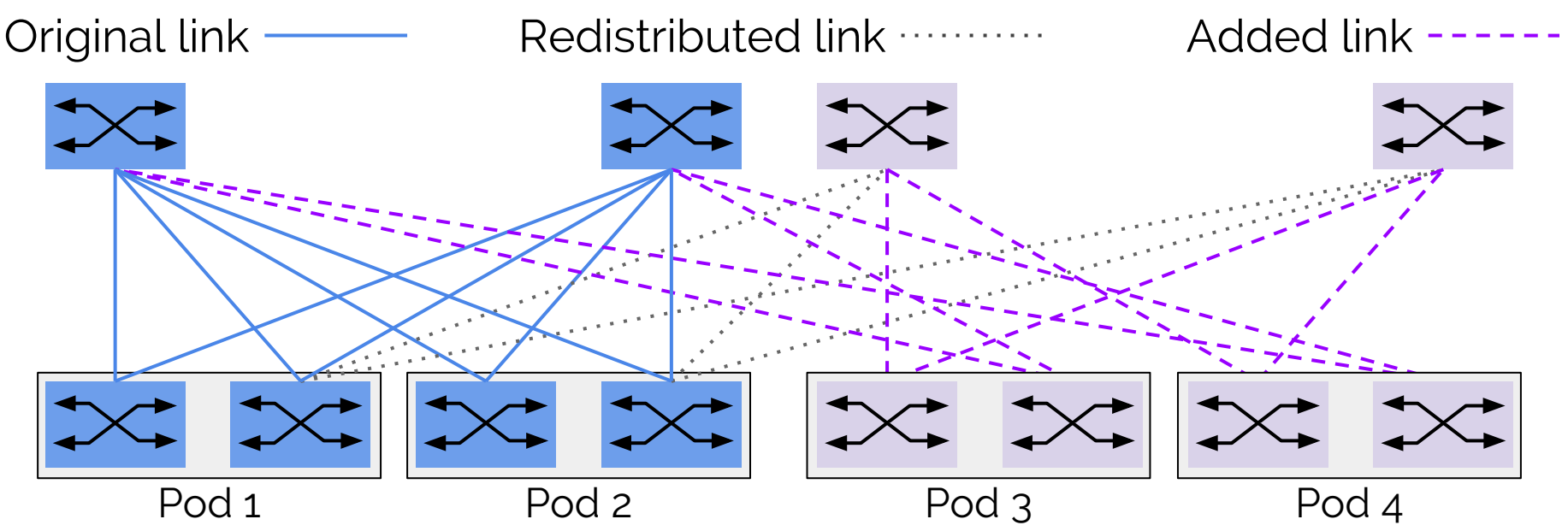
Large datacenters are **expanded incrementally**

Resources added as needed to reduce cost, depreciation

Challenges of Incremental Expansions

Expansions often require **rewiring existing connections**
Rewiring operations must be performed on ***live*** network

Example: Expanding a Fat-tree Topology



1) Add new pods and spines
2) Redistribute links from existing pods across spines
3) Connect new pods to spines

Example: Expanding a Fat-tree Topology

Is it possible to rewire this topology
without impacting production traffic?

Example: Expanding a Fat-tree Topology

Just rewire one cable at a time!

little risk, easy to plan, **but *slow***

Example: Expanding a Fat-tree Topology

Rewire multiple cables at a time!

faster, **but more risk, difficult to plan**

Rewire multiple cables at a time!

faster, **but more risk, difficult to plan**

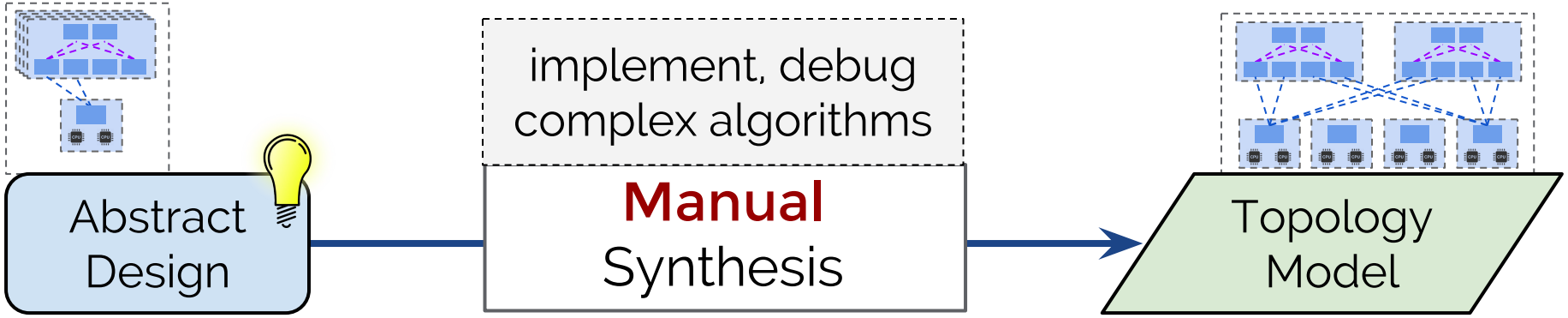
Which design is *better for SLO compliance*?

cannot evaluate via simple calculations

Architects need **good tools** to evaluate designs

But today, evaluating a design's utility is a **human-intensive process**

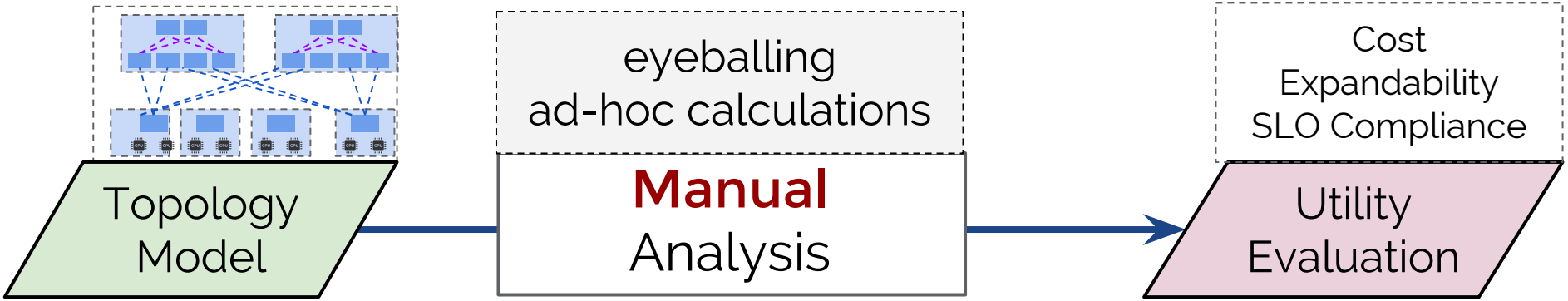
Today's Design Cycle: Abstract to Concrete Model



Generating a **concrete model** of wiring requires

implementing algorithms

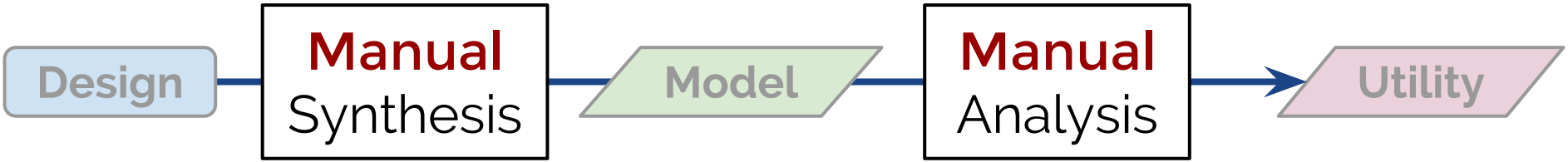
Today's Design Cycle: Concrete Model to Utility



Evaluating **utility** of a design is often done by

eyeballing and ad-hoc calculations

Today's **Human-Intensive** Design Cycle



This **human-intensive** design cycle
leads to **slow, cursory evaluations**

Slow, cursory evaluation = **big problem**

Slow, cursory evaluation = **big problem**

Because to find the ***best*** design architects need:

- **deep insight into design's utility**
- **to be able to quickly explore variants...**

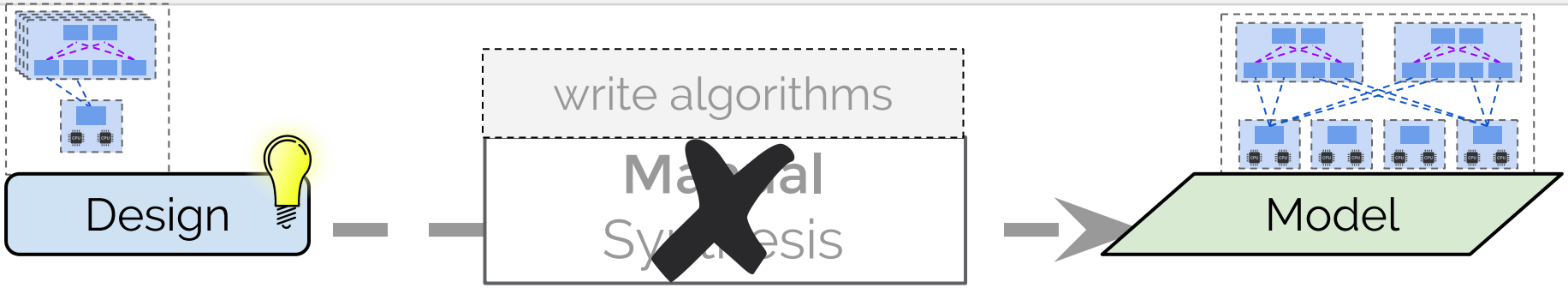
But if we had **better tools**

We could explore the design space **more efficiently**

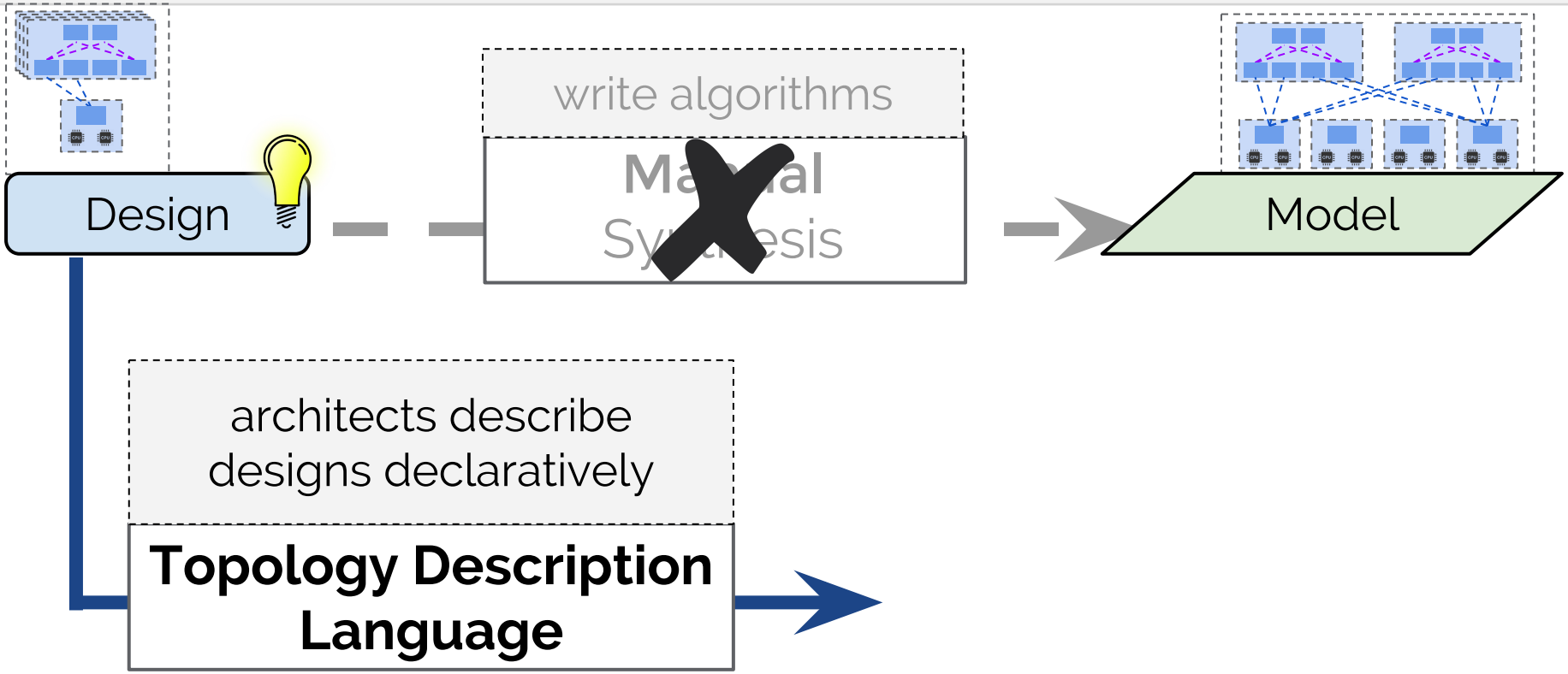
Condor

Enables rapid exploration of the design space

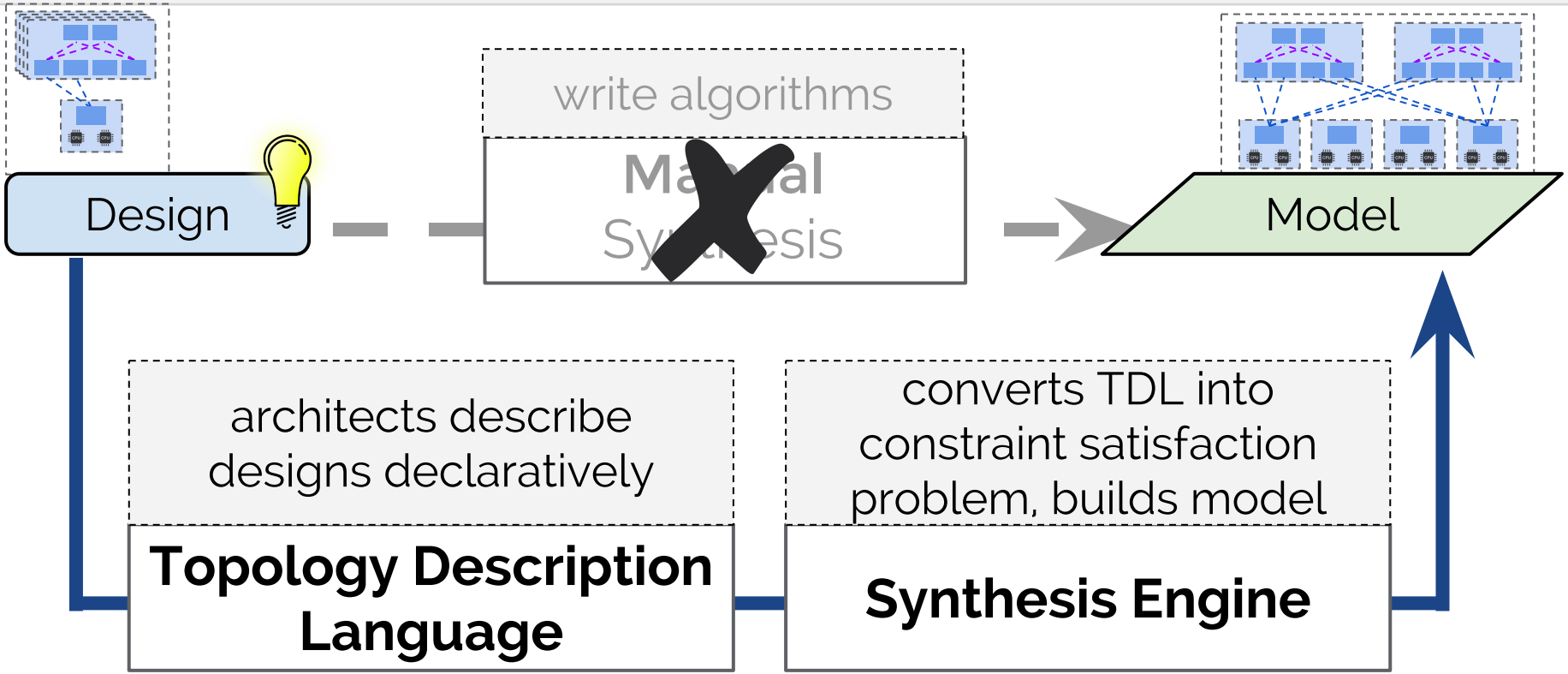
Condor's Design Cycle: Abstract to Concrete Model



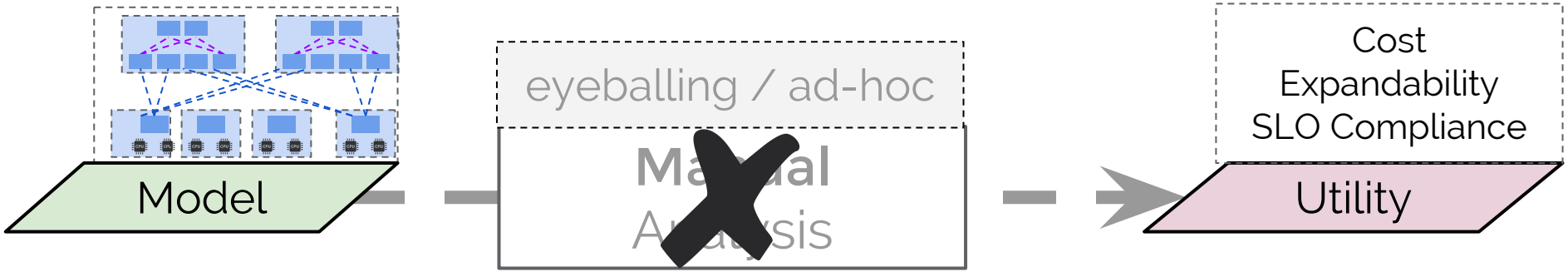
Condor's Design Cycle: Abstract to Concrete Model



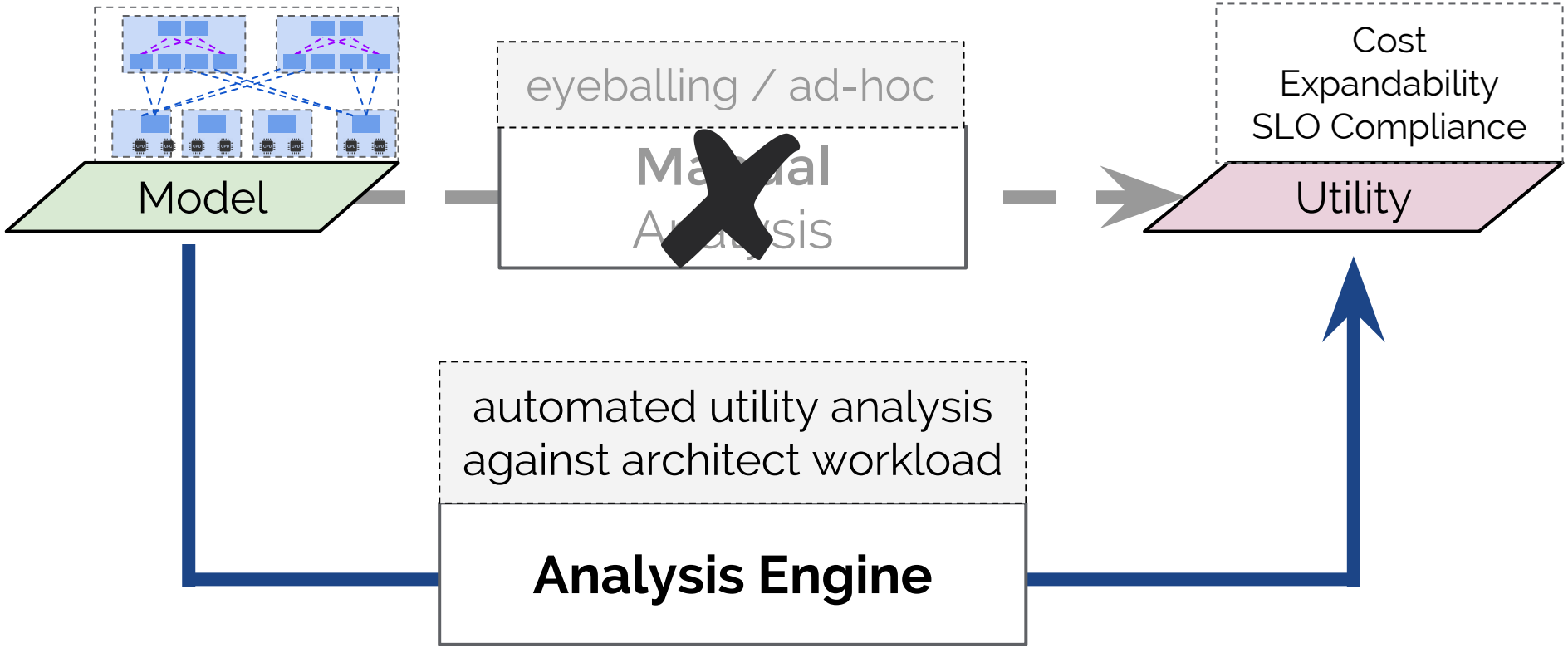
Condor's Design Cycle: Abstract to Concrete Model



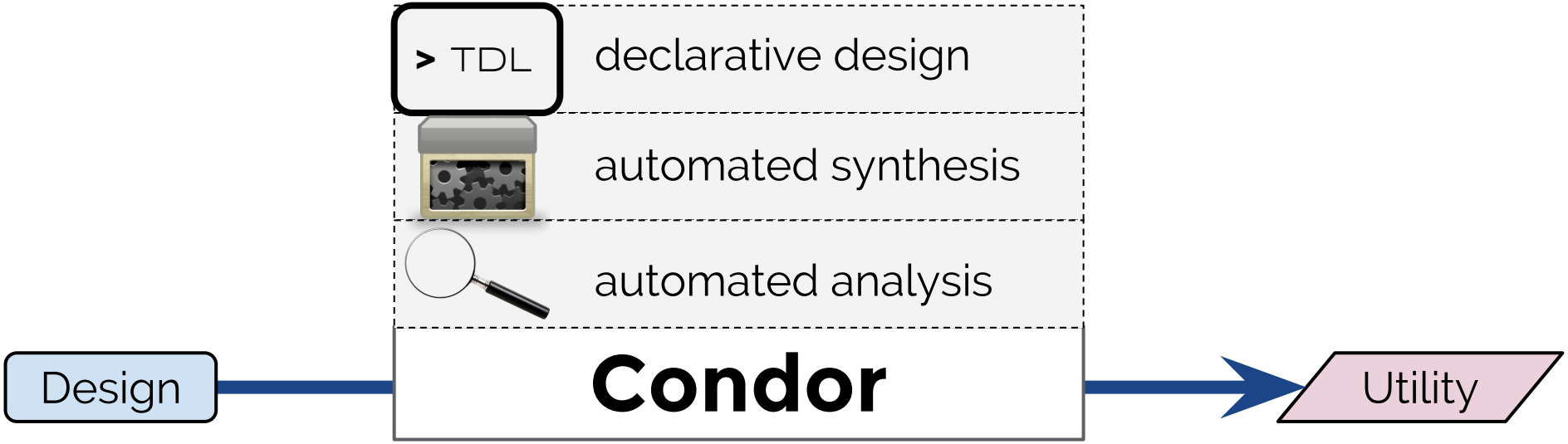
Condor's Design Cycle: Concrete Model to Utility



Condor's Design Cycle: Concrete Model to Utility



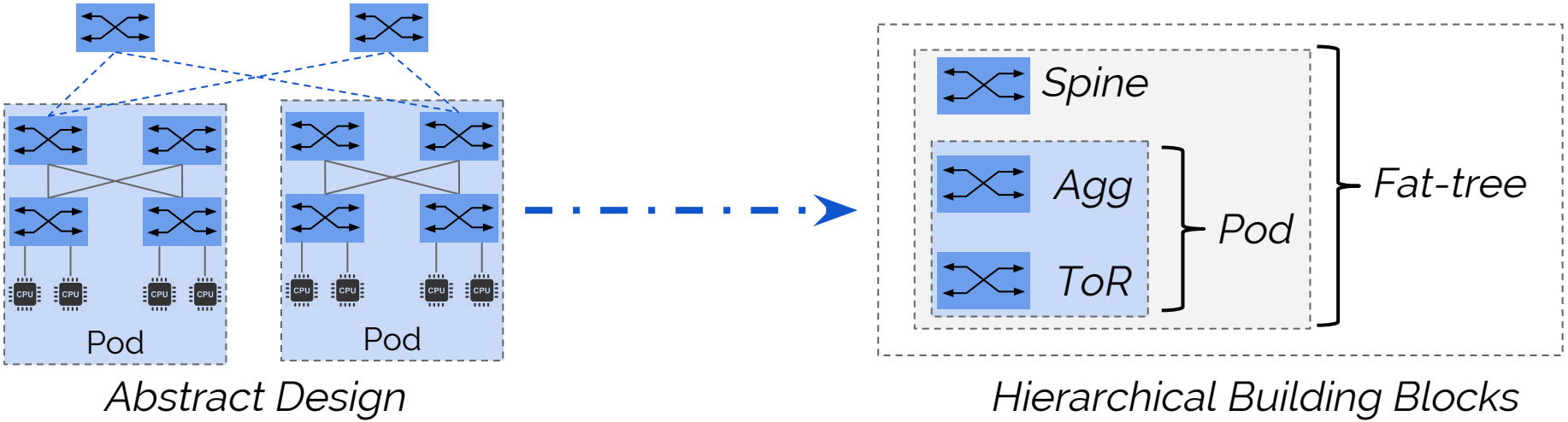
Condor Enables **Aggressive Exploration**



Condor pipeline enables
aggressive exploration of the design space

Designing a Fat-tree with **Condor's TDL**

Describing Fat-tree Topology with Condor's TDL



Abstract Design

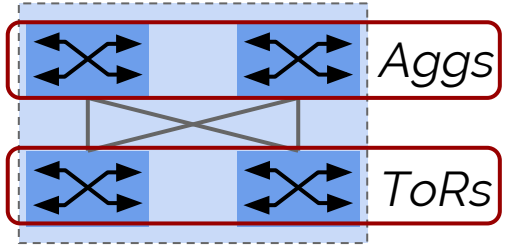
Hierarchical Building Blocks

Step 1: Define hierarchical building blocks

switches, linecards, racks, and parent-child relationships between them

Describing Fat-tree *Building Blocks* with Condor's TDL

(switches, linecards, racks, and the relationships between them)



Pod

```
class FatTree extends TDLBuildingBlock:
  agg = new Switch10GbE(num_ports, "agg")
  tor = new Switch10GbE(num_ports, "tor")

  pod = new TDLBuildingBlock("pod")
  pod.Contains(agg, num_sw_per_tier)
  pod.Contains(tor, num_sw_per_tier)
  ...
```

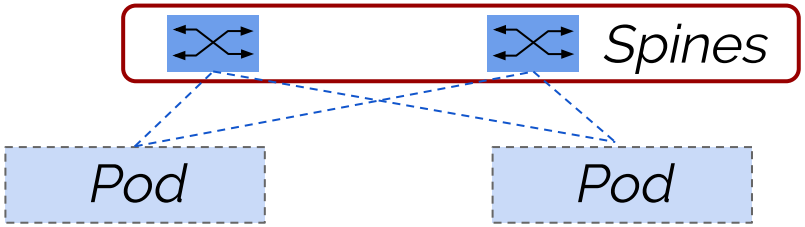
agg/ToR building blocks

pod building block

Fat-tree building block

Describing Fat-tree *Building Blocks* with Condor's TDL

(switches, linecards, racks, and the relationships between them)



Fat-tree

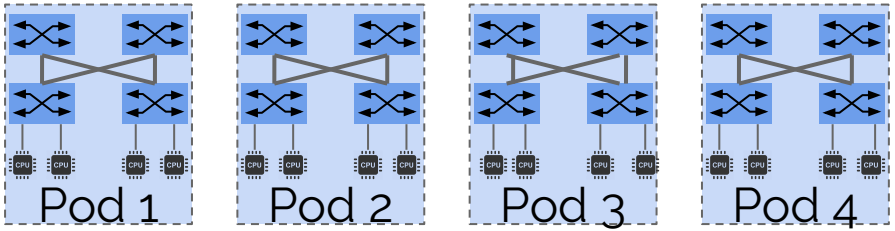
```
class FatTree extends TDLBuildingBlock:
   ...
   spine = new Switch10GbE(num_ports, "spine")

   Contains(spine, num_spines)
   Contains(pod, num_pods)
```

spine building blocks

Fat-tree contains spines and pods

Fat-tree building block

Describing Fat-tree Topology with Condor's TDL



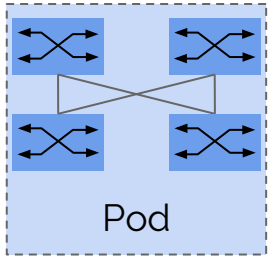
Each **pod** contains:
Servers
Top of Rack Switches
Aggregation Switches

```
agg = new Switch10GbE(...)
tor = new Switch10GbE(...)

pod = new Block("pod")
pod.Contains(agg, ...)
pod.Contains(tor, ...)
```

Building blocks are **naturally defined**

Describing FatTree *Connectivity* with Condor's TDL

(constraints on connectivity between components)



in every pod, every agg connects to every ToR

Abstract Design

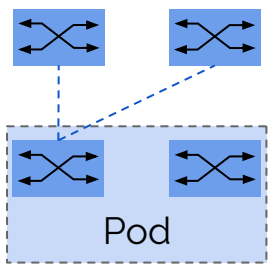
```
pod.ConnectPairsWithXLinks(agg, tor, 1)
```

TDL Constraint

Step 2: Define constraints on connectivity between building blocks

Describing FatTree *Connectivity* with Condor's TDL

(constraints on connectivity between components)



in every fat-tree, every spine connects to every pod

Abstract Design

`ConnectPairsWithXLinks``(spine, pod, 1)`

TDL Constraint

Step 2: Define constraints on connectivity between building blocks

```
pod.ConnectPairsWithXLinks(agg, tor, 1)
ConnectPairsWithXLinks(spine, pod, 1)
```

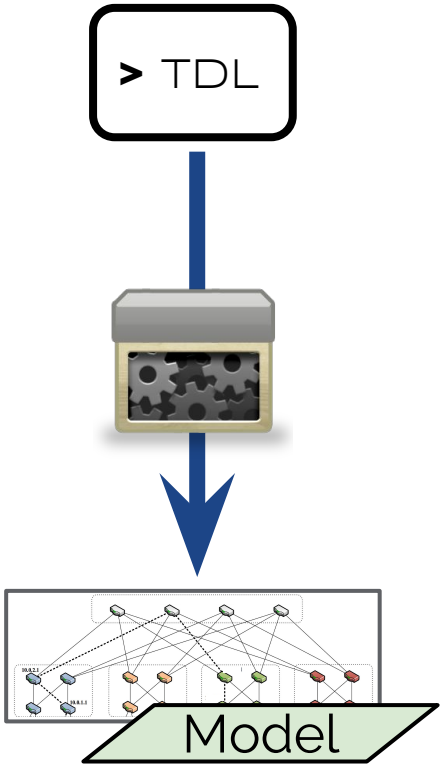
Done.

```
pod.ConnectPairsWithXLinks(agg, tor, 1)
ConnectPairsWithXLinks(spine, pod, 1)
```

Done.

Just ***two*** TDL constraints gives us a fat-tree's connectivity

Describing Fat-tree Topology with Condor's TDL



Fat-tree in < 15 lines of Condor's TDL

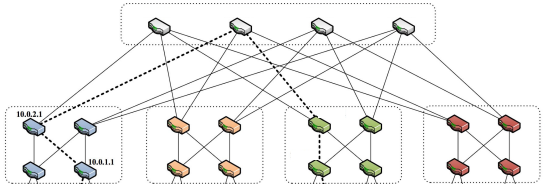
Condor's Automated Synthesizer

Concrete fat-tree topology

Benefit of Condor's TDL Constraints

```
__global__
void applyScaleFactorsKernel( uint8_T * const out,
    uint8_T const * const in, const double *factor,
    size_t const numRows, size_t const numCols )
{
    // Work out which pixel we are working on.
    const int rowIdx = blockIdx.x * blockDim.x + threadIdx.x;
    const int colIdx = blockIdx.y;
    const int sliceIdx = threadIdx.z;

    // Check this thread isn't off the image
    if( rowIdx >= numRows ) return;

    // Compute the index of my element
    size_t linearIdx = rowIdx + colIdx*numRows +
        sliceIdx*numRows*numCols;

    // Scale the value with the appropriate factor
    double result = (double)in[linearIdx]*factor[sliceIdx];

    // Cap our values to the interval [0,255]
    result = min(255.0,result);

    // Write the value to global memory.
    out[linearIdx] = (uint8_T)rint(result);
```

`Code` for Manual Synthesis



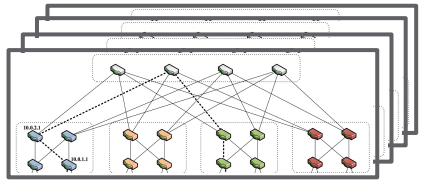
Single Design

If we had written `code` to build a fat-tree,
it could *only* generate a **single solution**

Benefit of Condor's TDL Constraints

> TDL

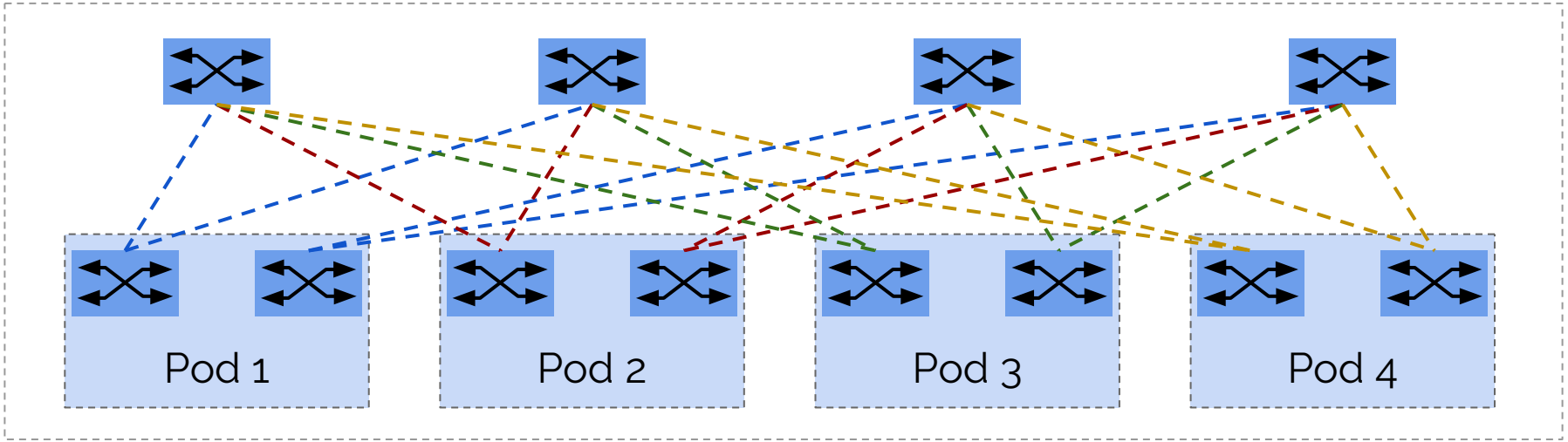
Fat-tree in Condor TDL



Solution Space

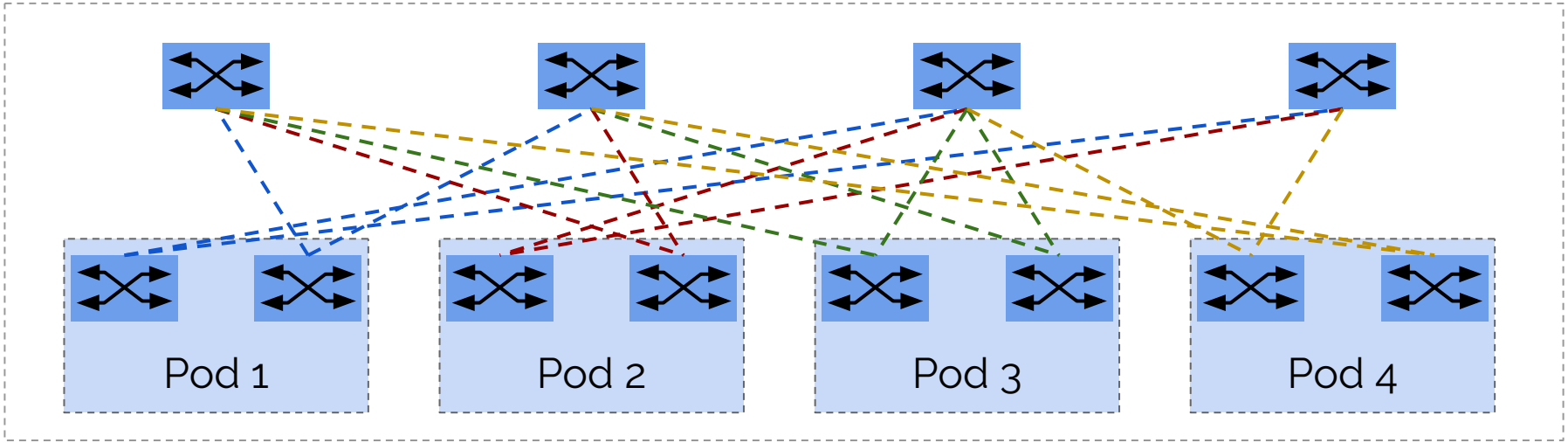
But the constraints we defined with TDL can be satisfied by **multiple solutions**

Benefit of Condor's TDL Constraints



The canonical wiring meets the constraints…

Benefit of Condor's TDL Constraints



This wiring also meets the constraints…

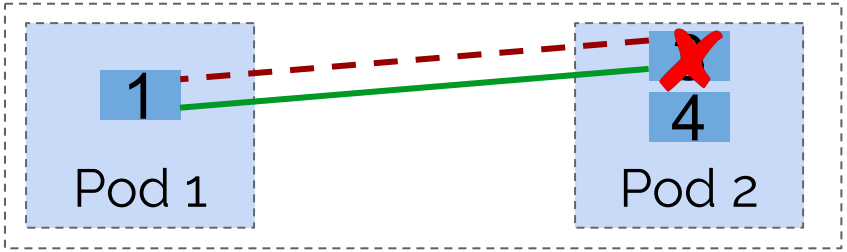
And **F10** meets these constraints

To describe **F10** with Condor's TDL,
we **add constraints** to reduce the solution space

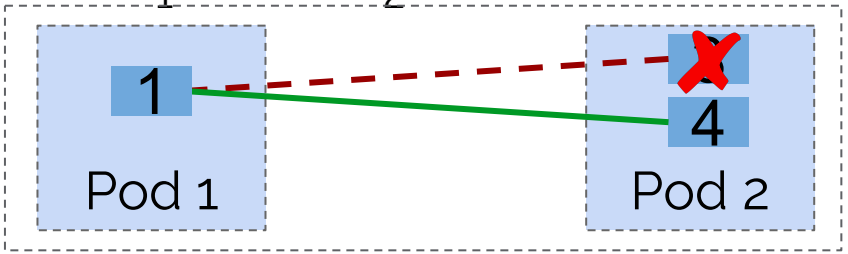
Describing F10 with Condor's TDL

Recall: F10 increases inter-pod path diversity

e.g.: # of paths from agg_1 to pod_2



Standard wiring = **one path**
(via agg_3)

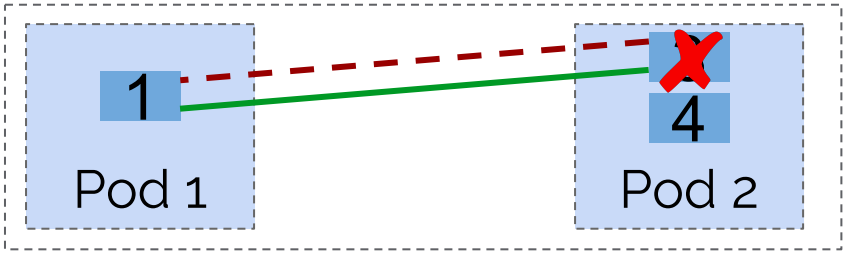


F10's wiring = **two paths**
(via agg_3 and agg_4)

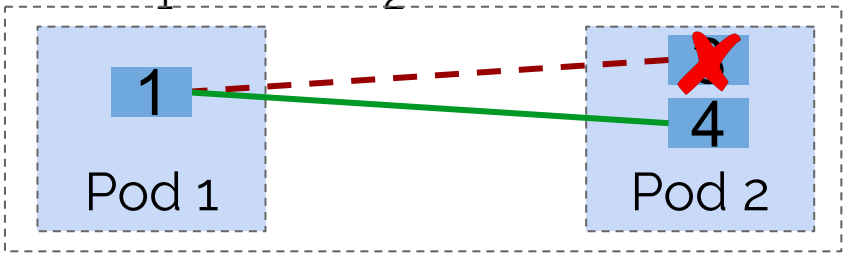
Describing F10 with Condor's TDL

Recall: F10 increases inter-pod path diversity

e.g.: # of paths from agg_1 to pod_2



Standard wiring = **one path**
(via agg_3)



F10's wiring = **two paths**
(via agg_3 and agg_4)

Let's try using the following constraint:

Connect every agg sw to as many other agg sw as possible

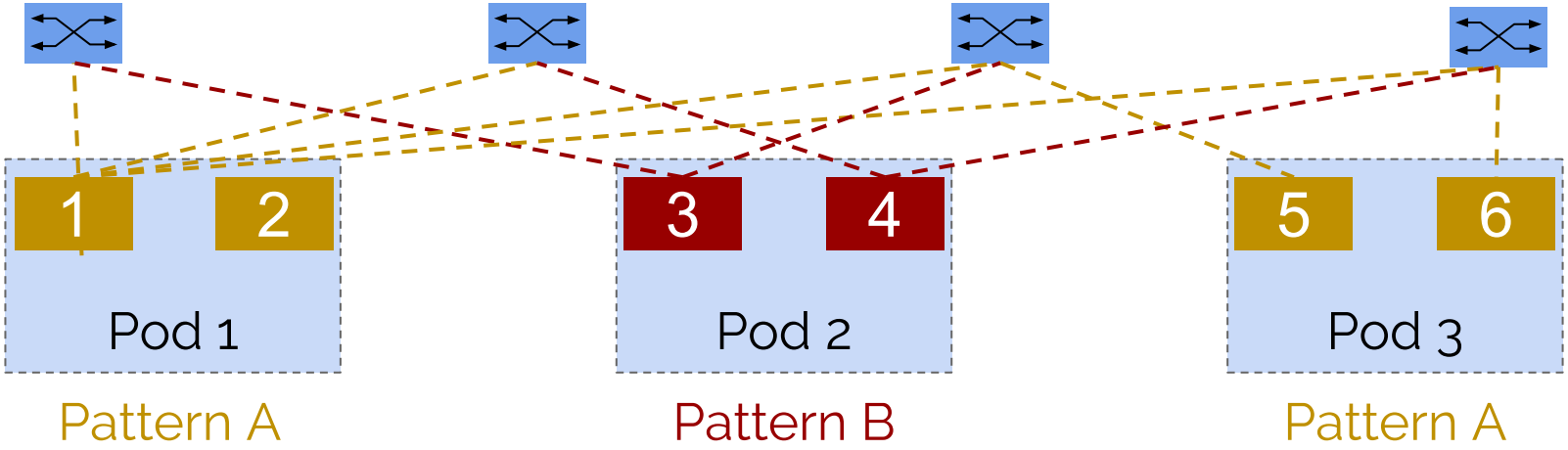
We used this constraint to try to ***build*** F10

We used this constraint to try to ***build*** F10

But instead we ended up ***improving*** F10

Condor found **better solutions** than F10's algorithm

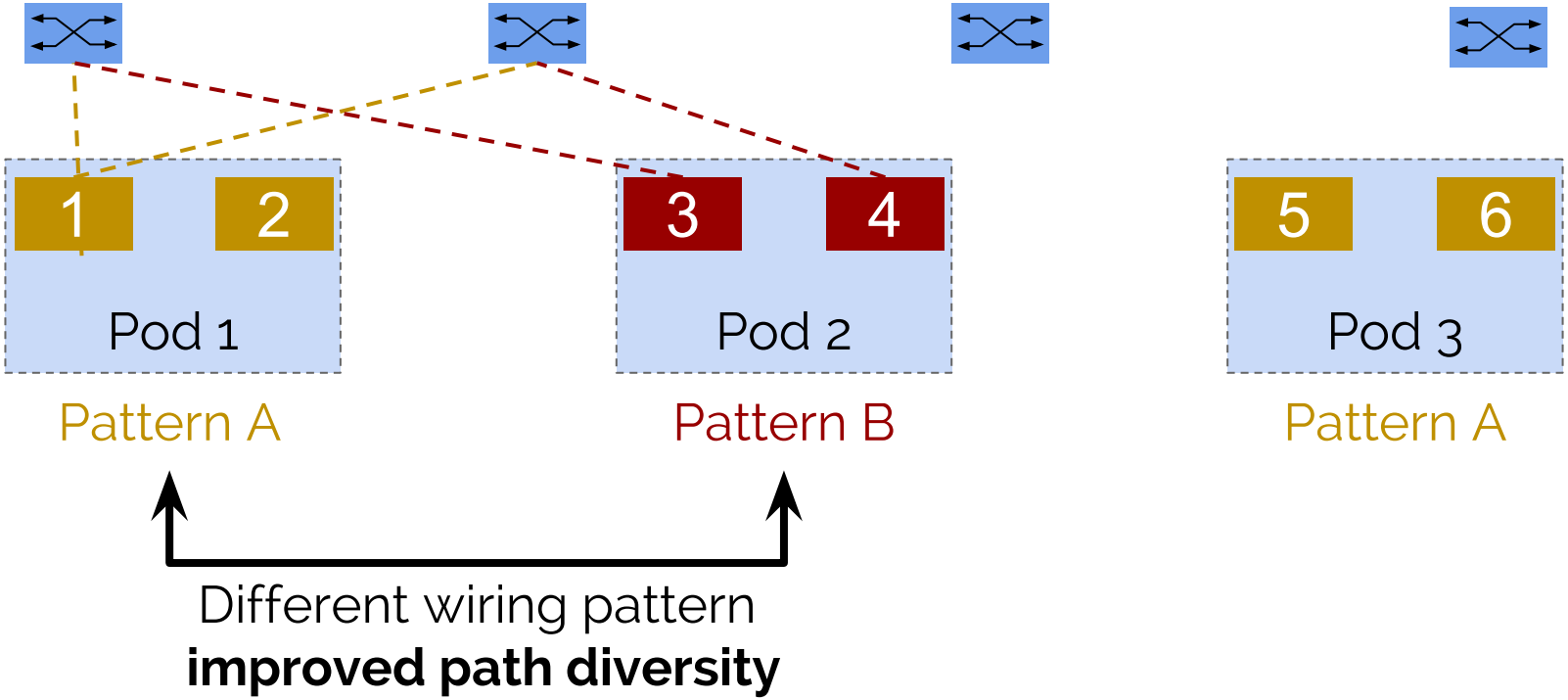
Limitations of F10's Imperative Algorithm



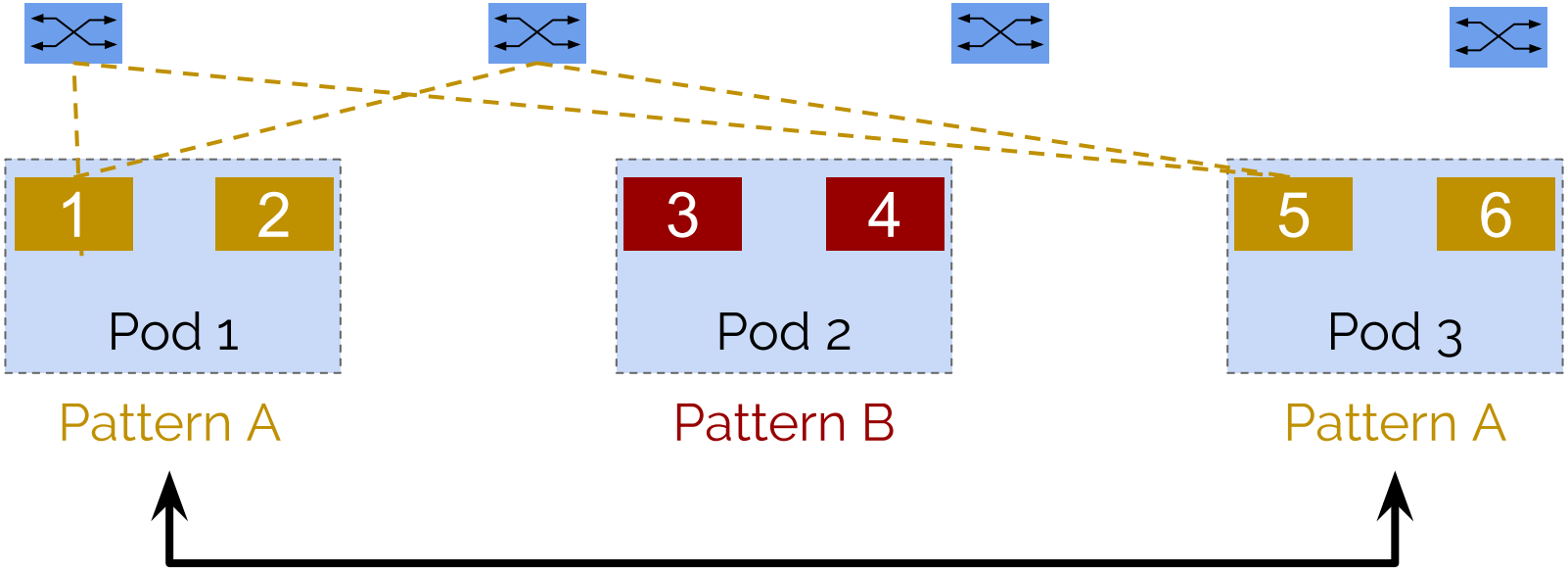
F10 generates and assigns two patterns of wiring: *A* and *B*

Only improves connectivity between pods with different patterns

Limitations of F10's Imperative Algorithm



Limitations of F10's Imperative Algorithm

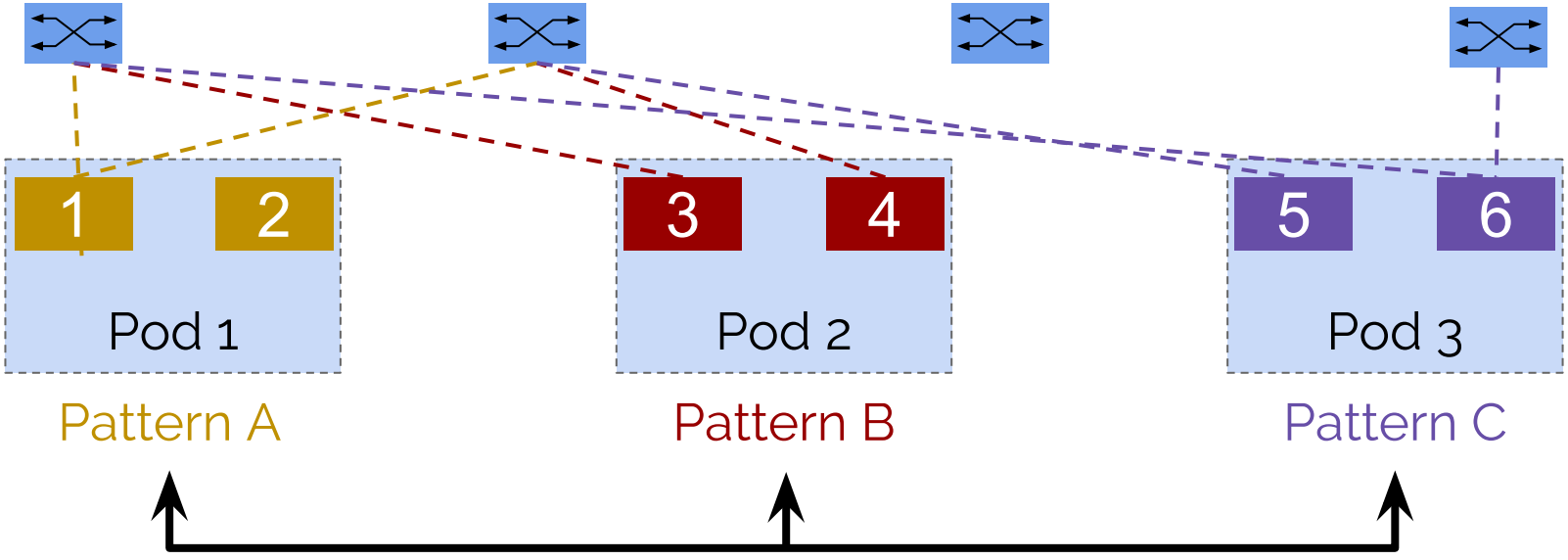


Path diversity is limited by the number of wiring patterns
and F10's algorithm only produces two patterns

Path diversity is limited by the number of wiring patterns
and F10's algorithm only produces two patterns

Probability that a pair of pods has better path diversity is
limited to 50%

Improving F10 with Condor



Condor gives three wiring patterns =
maximum path diversity for 3 pod fat-tree

Improving F10 with Condor

F10 Algorithm

# of pods	# of patterns	probability of path diversity
2	**2**	**50%**

Condor's Solution

# of pods	# of patterns	probability of path diversity
2	**2**	**50%**

And with our TDL constraint + Condor's synthesizer

path diversity increases as the topology grows

Improving F10 with Condor

F10 Algorithm

# of pods	# of patterns	probability of path diversity
2	2	50%
4	**2**	**50%**

Condor's Solution

# of pods	# of patterns	probability of path diversity
2	2	50%
4	**3**	**66%**

And with our TDL constraint + Condor's synthesizer

path diversity increases as the topology grows

Improving F10 with Condor

F10 Algorithm

# of pods	# of patterns	probability of path diversity
2	2	50%
4	2	50%
16	**2**	**50%**

Condor's Solution

# of pods	# of patterns	probability of path diversity
2	2	50%
4	3	66%
16	**9**	**88%**

And with our TDL constraint + Condor's synthesizer

path diversity increases as the topology grows

With Condor, **we improved F10**
by **systematically exploring**
the design space

Could we have found these wirings by writing **an algorithm**?

Could we have found these wirings by writing **an algorithm**?

Very unlikely.

These patterns are instances of Balanced Incomplete Block Design (BIBD)

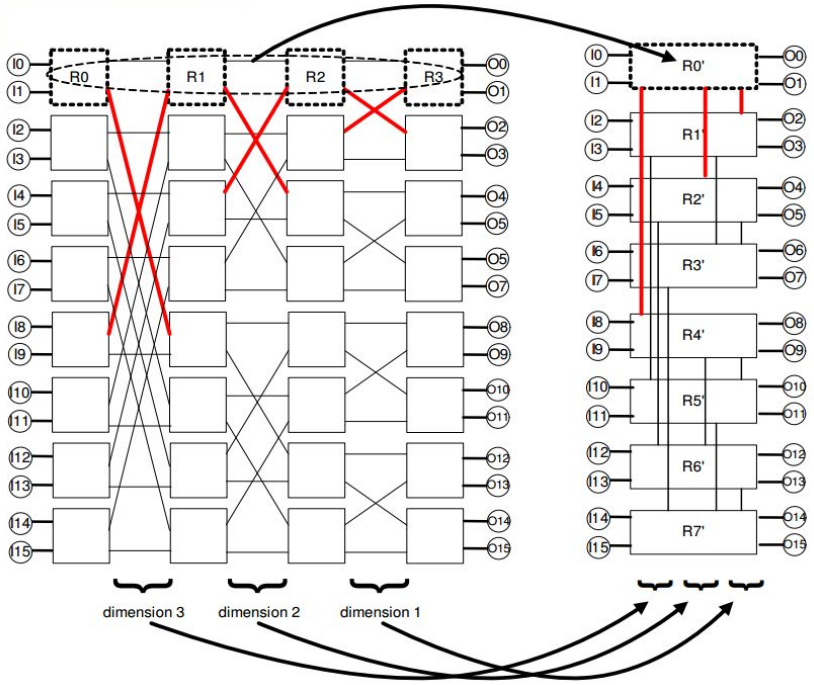
Why does BIBD matter?

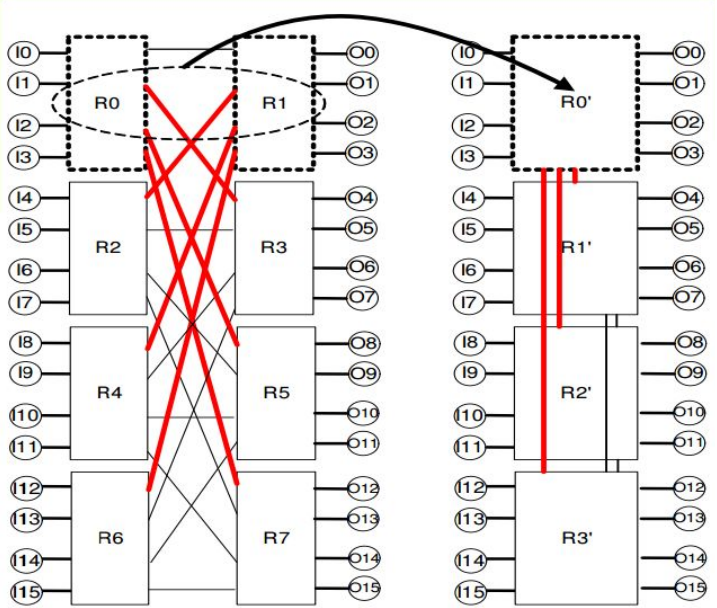
1) Synthesis of BIBDs is known to be difficult
2) We didn't know about BIBD until *after* finding these solutions
3) **Unlikely that architect alone could come up with these designs**

What Topologies Does Condor's TDL Support?

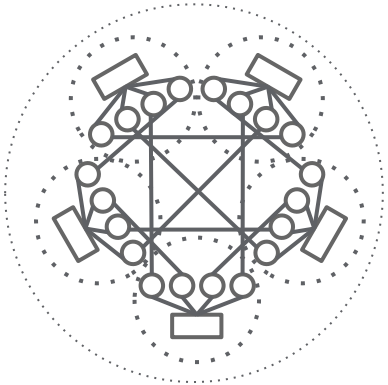
TDL supports *any* topology that you can describe with
building blocks and *connectivity constraints*

Condor's TDL supports **flattened-butterfly topologies**

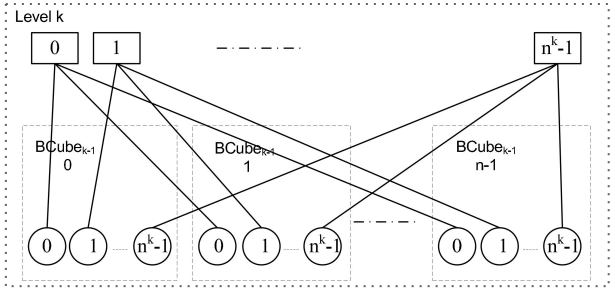




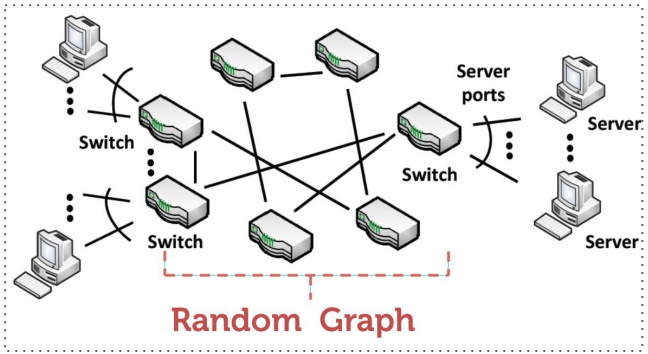
TDL supports **recursive** and **random-graph topologies**



DCell



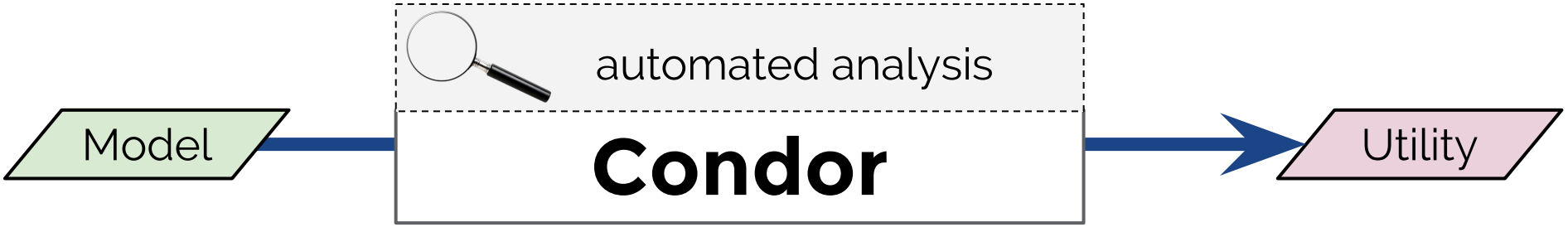
BCube



Jellyfish

And Condor synthesizes topologies quickly…

Fat-tree with 128k hosts in < 2 minutes
DCell with 360k hosts in < 6 minutes



After we've described and built a design

Condor can help us evaluate **its utility**

Design's **utility** ≠ how it performs in worst-case

In large networks, worst-case scenarios are unlikely

Design's **utility** ≠ how it performs in worst-case

In large networks, worst-case scenarios are unlikely

Traditional worst-case metrics less useful

~~bisection bandwidth~~

~~potential for partitions~~

Instead, to evaluate a design's **utility**, we focus on

SLO Compliance

Particularly if it can carry an **application's traffic**

(e.g.: ≥ 100 Mbps **throughput** 99.99% of the time)

SLO Compliance is Evaluated over a

Topology's Lifecycle

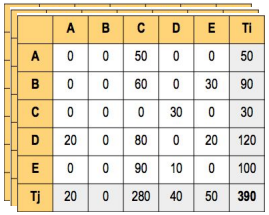
SLO Compliance is Evaluated over a

Topology's Lifecycle

During **failures** and **expansions**
will this topology continue to meet its SLO?

Estimating SLO Compliance: Architect Inputs

Architect characterizes **workload** as **traffic matrixes**

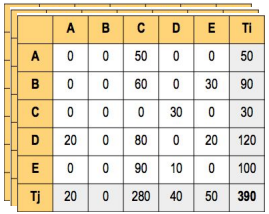


	A	B	C	D	E	Ti
A	0	0	50	0	0	50
B	0	0	60	0	30	90
C	0	0	0	30	0	30
D	20	0	80	0	20	120
E	0	0	90	10	0	100
Tj	20	0	280	40	50	**390**

Workload
(Traffic Matrixes)

Estimating SLO Compliance: Architect Inputs

Architect characterizes **reliability** as **failure and recovery rates**



	A	B	C	D	E	Ti
A	0	0	50	0	0	50
B	0	0	60	0	30	90
C	0	0	0	30	0	30
D	20	0	80	0	20	120
E	0	0	90	10	0	100
Tj	20	0	280	40	50	**390**

Workload
(Traffic Matrixes)

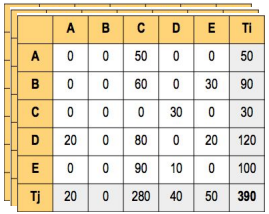
+

MTBF + MTTR
> TDL

Reliability Info

Estimating SLO Compliance: Architect Inputs

Architect characterizes **expansions** as **rewiring / recabling plan**



	A	B	C	D	E	Ti
A	0	0	50	0	0	50
B	0	0	60	0	30	90
C	0	0	0	30	0	30
D	20	0	80	0	20	120
E	0	0	90	10	0	100
Tj	20	0	280	40	50	**390**

Workload
(Traffic Matrixes)

+

MTBF + MTTR
> TDL

Reliability Info

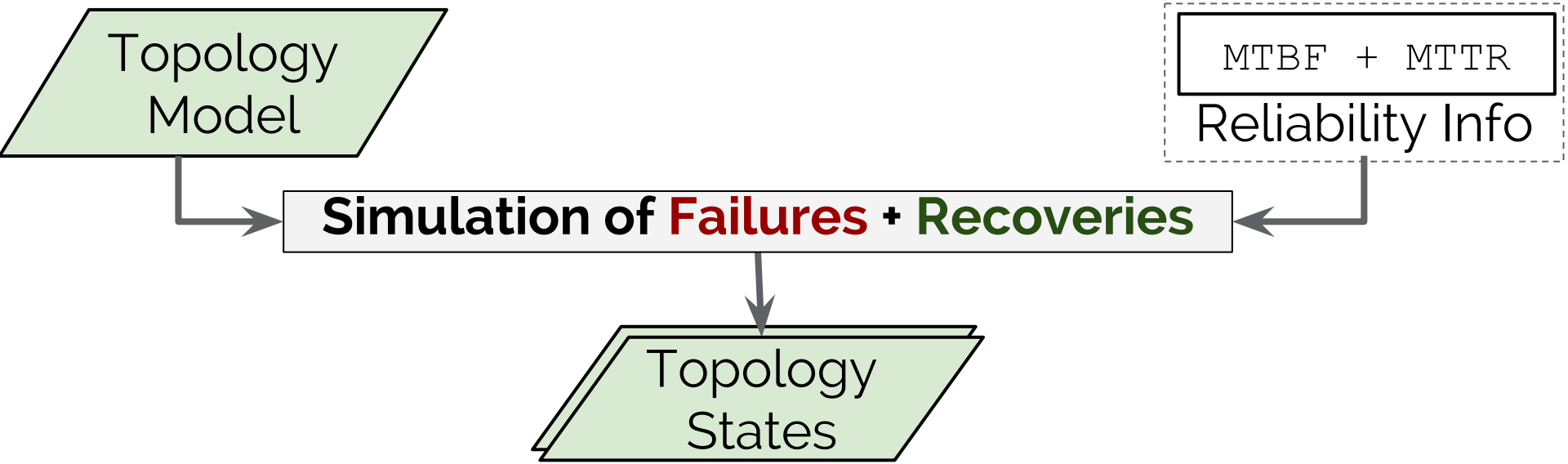
+

Procedure for adding / changing cabling

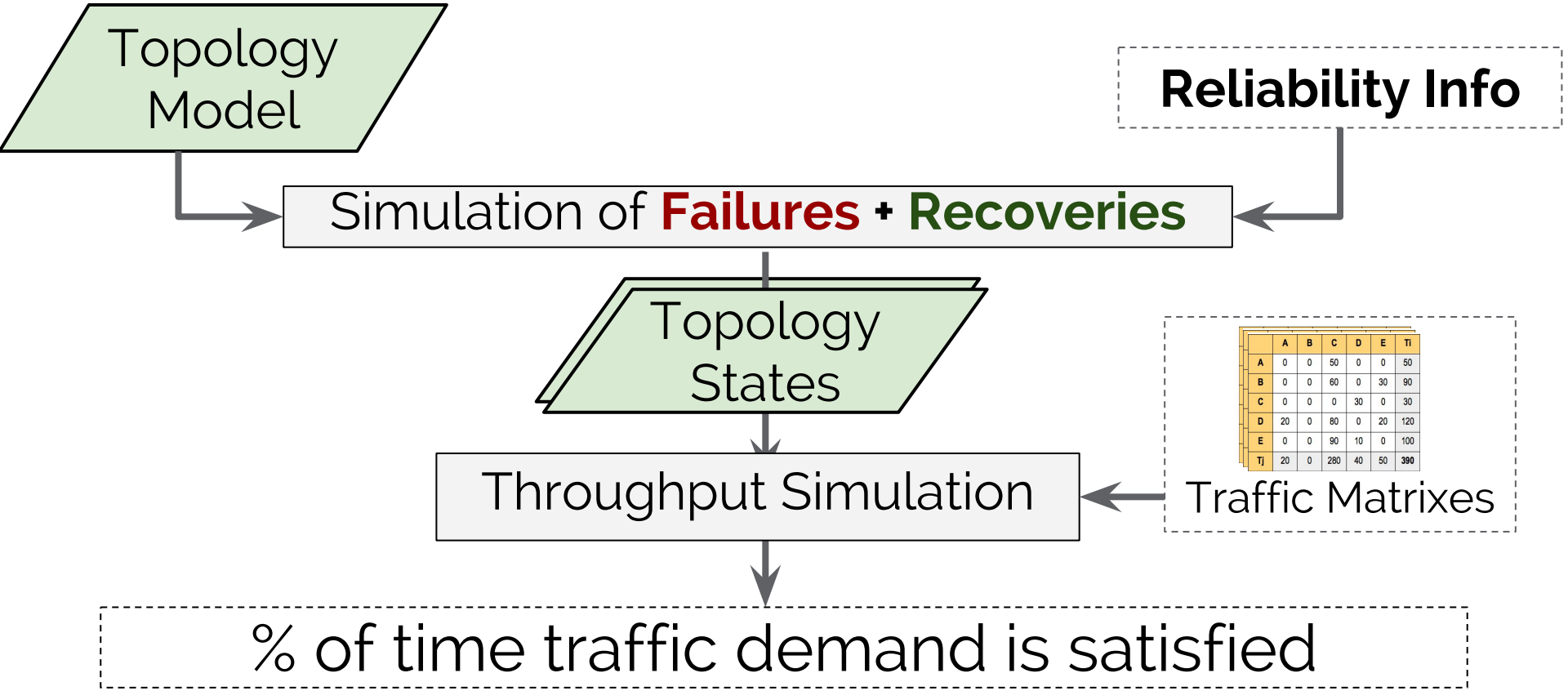
Rewiring Plan

Estimating SLO Compliance: Failures

During **failures**
will this topology continue to meet its SLO?



Estimating SLO Compliance: Failures



	A	B	C	D	E	Ti
A	0	0	50	0	0	50
B	0	0	60	0	30	90
C	0	0	0	30	0	30
D	20	0	80	0	20	120
E	0	0	90	10	0	100
Tj	20	0	280	40	50	390

During **expansions**
will this topology continue to meet its SLO?

Almost **identical** process

Estimating SLO Compliance: Expansions

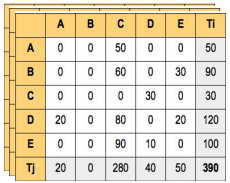
Topology Model

Rewiring Plan

Simulation of **Expansion Operations**

Topology States

Throughput Simulation



	A	B	C	D	E	Ti
A	0	0	50	0	0	50
B	0	0	60	0	30	90
C	0	0	0	30	0	30
D	20	0	80	0	20	120
E	0	0	90	10	0	100
Tj	20	0	280	40	50	390

Traffic Matrixes

% of time traffic demand is satisfied

Evaluating Online Expandability with Condor

Use Condor to evaluate tree expansion

Max size = ~50k servers

Each increment = +6k servers

Rewiring = 32 operations per increment

Simulate 225 stage expansion operation

(difficult without Condor)

Check if design remained SLO compliant

In the Paper Example of Expansions
Expandability of two designs with a minor variation in TDL

Condor helped us evaluate SLO compliance

over the lifecycle of these topologies

Conclusion: The Condor Approach

Condor replaces a **human-intensive design process**
with a **systematic approach to topology design**

Condor's TDL makes it easier to design topologies
Condor's synthesizer quickly models topologies from TDL
Condor's analysis engine provides insight into a design's utility

Conclusion: The Condor Approach

Condor replaces a **human-intensive design process**
with a **systematic approach to topology design**

More Examples in Paper:
Evaluating SLO compliance of different designs
Other trade-offs in expansions (e.g.: imbalance in routing protocols)

TDL Source Code:
http://nsl.cs.usc.edu/condor