Introduction to DÆDÆLUS

Please ask questions about anything you don't understand

Welcome Dean V1.0 (02025-May-20)

How can you help?

- What is your learning process of the protocol? Help us learn how to teach it.
- What attracts you to DÆDÆLUS so far?
- Help build our product (Bits on the Wire Toolset for the FPGAs)?
- Help model our Algorithms Mathematically

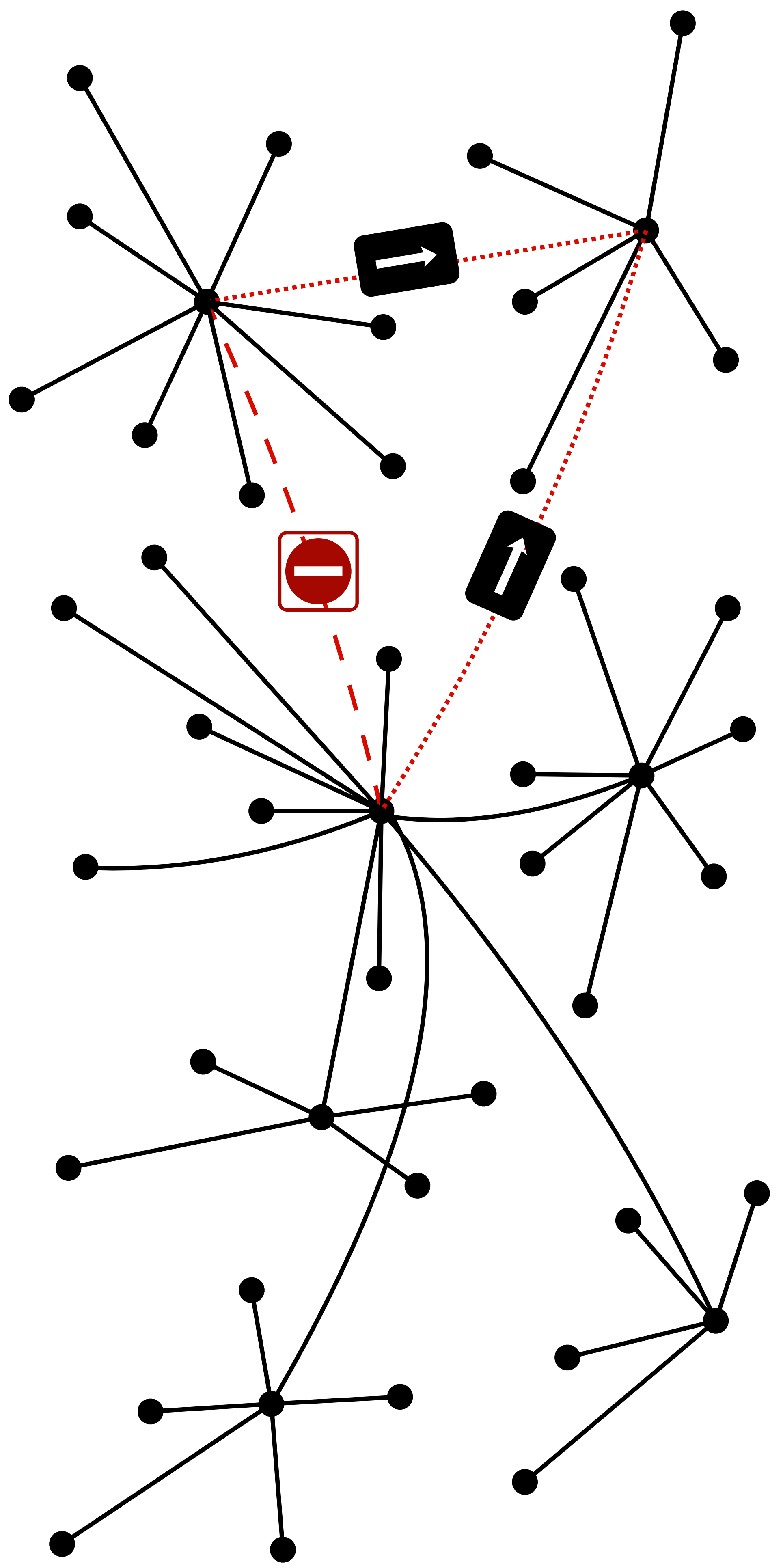
Problem

Maintaining liveness and synchronizing processes in networks is a challenge when packets can be dropped, reordered, duplicated or delayed

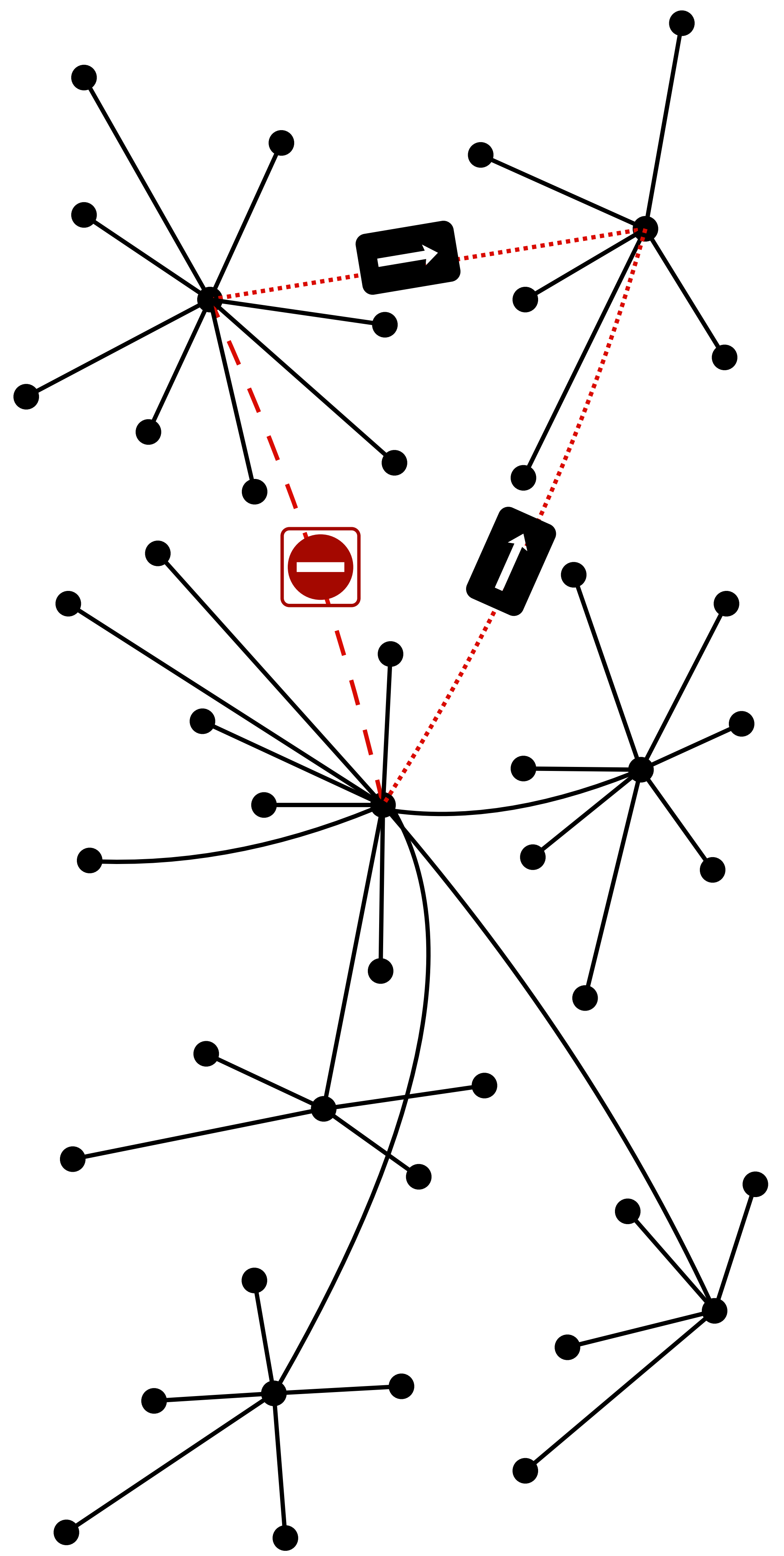
- Conventional networks require protocol stacks and applications to use timeouts and retries to maintain liveness
- This makes exactly-once semantics *impossible* and precipitates retry storms which lead to unbounded tail latency and transaction failure
- This results in silent corruption of data structures and loss of data.
  A problem seen in every distributed database for decades
- These failures are not understood because customers (under NDA) may not publish results that embarrass vendors

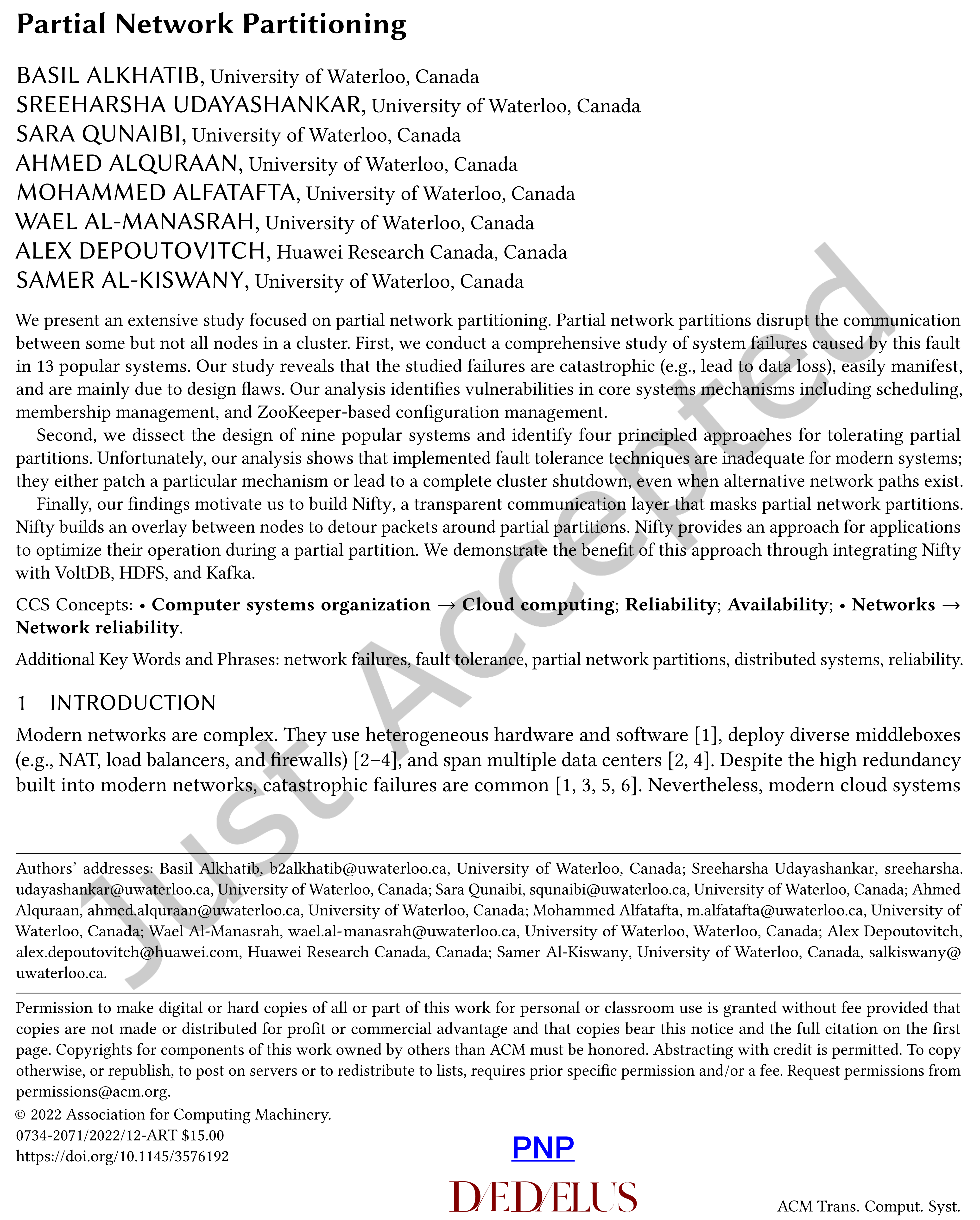
Network Faults Lead to Transaction Failure

- 80% of failures have a **catastrophic impact**, with data loss being the most common (27%)
- 90% of the **failures are silent**, the rest produce warnings that are unclear
- 21% of the failures lead to permanent damage to the system. This **damage persists** even after the network partition heals



Databases Fail with Existing Networks





Partial Network Partitioning

BASIL ALKHATIB, University of Waterloo, Canada
SREEHARSHA UDAYASHANKAR, University of Waterloo, Canada
SARA QUNAIBI, University of Waterloo, Canada
AHMED ALQURAAN, University of Waterloo, Canada
MOHAMMED ALFATAFTA, University of Waterloo, Canada
WAEL AL-MANASRAH, University of Waterloo, Canada
ALEX DEPOUTOVITCH, Huawei Research Canada, Canada
SAMER AL-KISWANY, University of Waterloo, Canada

We present an extensive study focused on partial network partitioning. Partial network partitions disrupt the communication between some but not all nodes in a cluster. First, we conduct a comprehensive study of system failures caused by this fault in 13 popular systems. Our study reveals that the studied failures are catastrophic (e.g., lead to data loss), easily manifest, and are mainly due to design flaws. Our analysis identifies vulnerabilities in core systems mechanisms including scheduling, membership management, and ZooKeeper-based configuration management.

Second, we dissect the design of nine popular systems and identify four principled approaches for tolerating partial partitions. Unfortunately, our analysis shows that implemented fault tolerance techniques are inadequate for modern systems; they either patch a particular mechanism or lead to a complete cluster shutdown, even when alternative network paths exist.

Finally, our findings motivate us to build Nifty, a transparent communication layer that masks partial network partitions. Nifty builds an overlay between nodes to detour packets around partial partitions. Nifty provides an approach for applications to optimize their operation during a partial partition. We demonstrate the benefit of this approach through integrating Nifty with VoltDB, HDFS, and Kafka.

CCS Concepts: • **Computer systems organization** → **Cloud computing**; **Reliability**; **Availability**; • **Networks** → **Network reliability**.

Additional Key Words and Phrases: network failures, fault tolerance, partial network partitions, distributed systems, reliability.

1 INTRODUCTION

Modern networks are complex. They use heterogeneous hardware and software [1], deploy diverse middleboxes (e.g., NAT, load balancers, and firewalls) [2–4], and span multiple data centers [2, 4]. Despite the high redundancy built into modern networks, catastrophic failures are common [1, 3, 5, 6]. Nevertheless, modern cloud systems

Authors' addresses: Basil Alkhatib, b2alkhatib@uwaterloo.ca, University of Waterloo, Canada; Sreeharsha Udayashankar, sreeharsha.udayashankar@uwaterloo.ca, University of Waterloo, Canada; Sara Qunaibi, squnaibi@uwaterloo.ca, University of Waterloo, Canada; Ahmed Alquraan, ahmed.alquraan@uwaterloo.ca, University of Waterloo, Canada; Mohammed Alfatafta, m.alfatafta@uwaterloo.ca, University of Waterloo, Canada; Wael Al-Manasrah, wael.al-manasrah@uwaterloo.ca, University of Waterloo, Waterloo, Canada; Alex Depoutovitch, alex.depoutovitch@huawei.com, Huawei Research Canada, Canada; Samer Al-Kiswany, University of Waterloo, Canada, salkiswany@uwaterloo.ca.

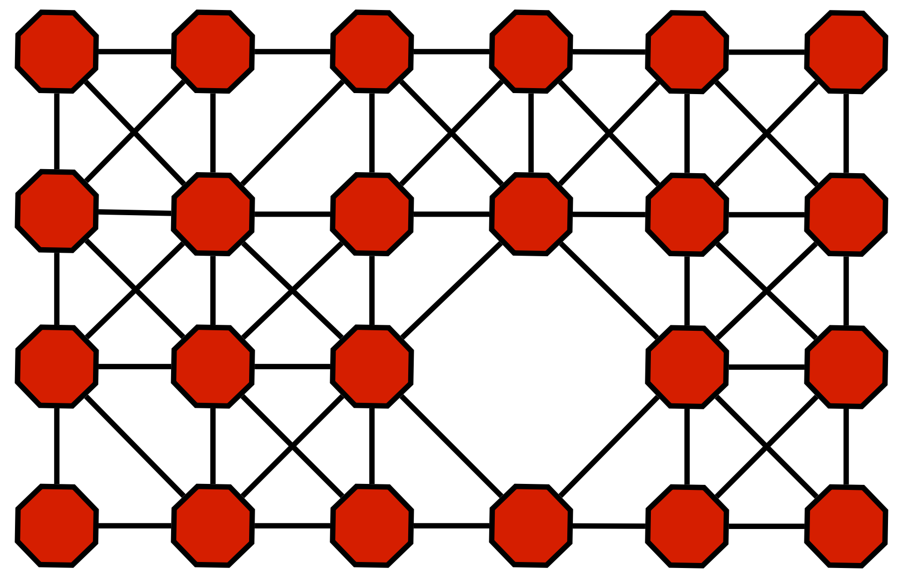
0734-2071/2022/12-ART $15.00
https://doi.org/10.1145/3576192

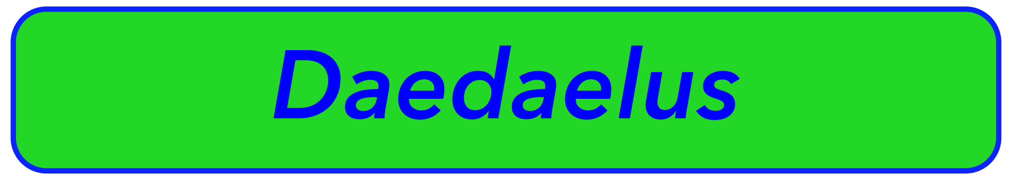
PNP

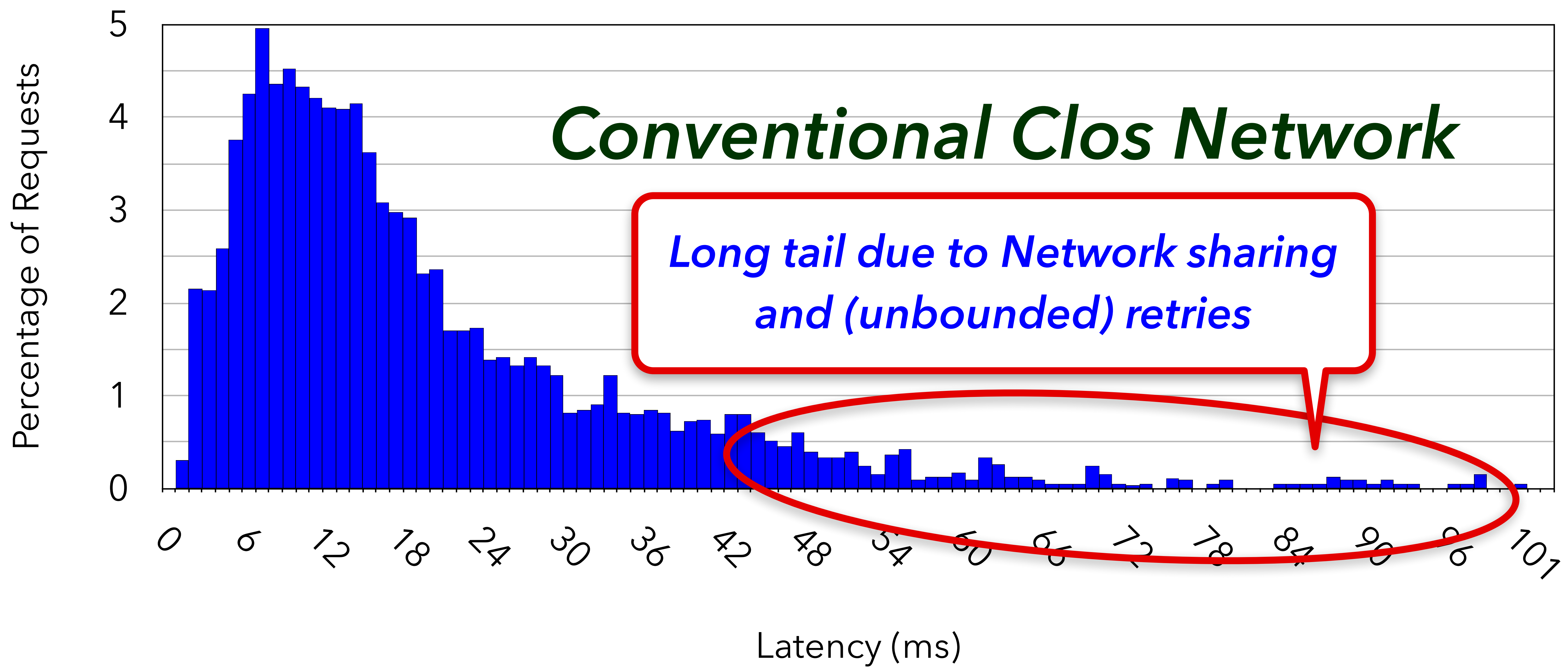
Tail Latency

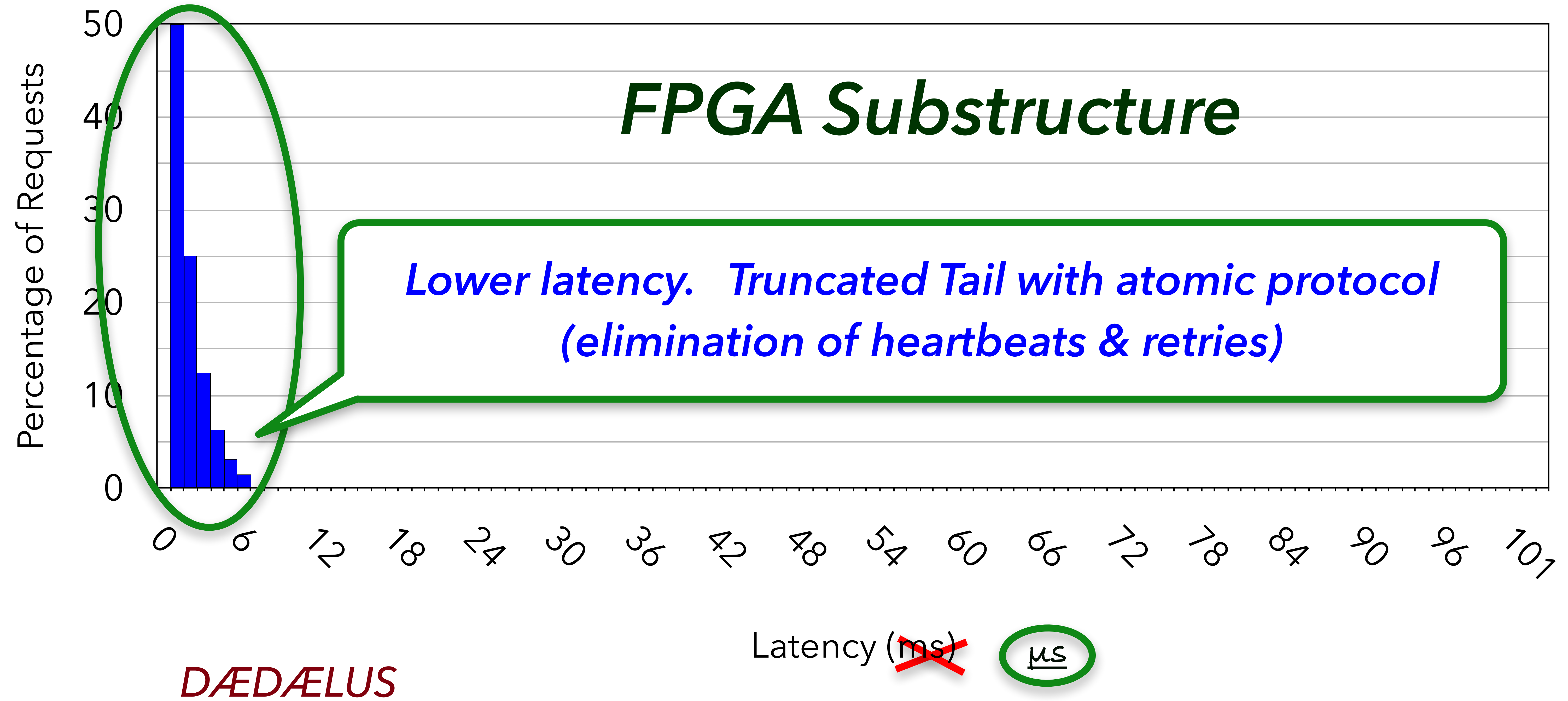
Daedaelus reduces latency in ways conventional networks cannot:

- Direct connections
- Multicast consensus, in parallel over 8 ports instead of serial over 1
- Truncated Tail Latency – protocol knows it failed or succeeded (without heartbeats or timeouts)









DÆ

DÆDÆLUS

Mission

Daedaelus: Architecting the Future with Enduring Excellence

We address fundamental problems in distributed systems using protocols, data structures and algorithms inspired by Quantum Information Theory and Multiway Systems.

Our market is secure, reliable, distributed computing on the edge. Use-cases include Transaction Processing, Digital Twins, Multiplayer Games & Interface to Quantum Computers

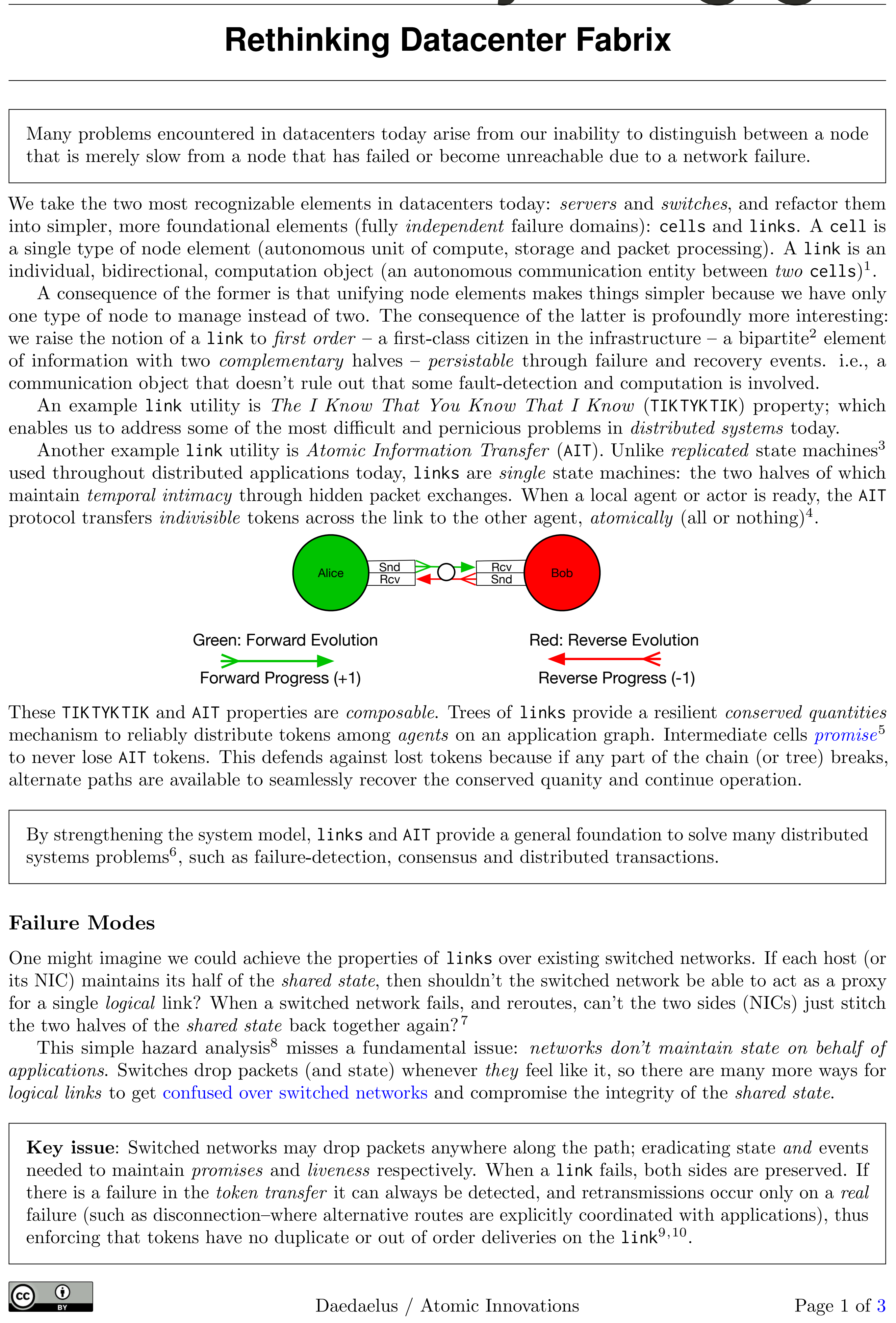
- **Applied Graph Theory**
- Digital Twins
- Ethernet & Other Chiplet Interconnects (UCIe)
- Distributed infrastructures where:
- 'Time-reversible' constructors (specified as Petri nets) no longer rely on the irreversible smash and restart of Shannon information to recover from failures, or to reconfigure computational resources.
- 'Precise Information-theoretic' emulators to enable competitive interactions for 'Digital Twins'.
- 'Rulesets' to enable non-zero-sum game (NZSG) outcomes, where winners are those who exhibit cooperative (trustable) behavior, and cheaters cannot hide behind a veil of anonymity or misinformation.

Shared on iCloud

Read Me First

In Slack

- Don’t let anything go by you don’t understand at this stage. Ask questions!



Rethinking Datacenter Fabrix

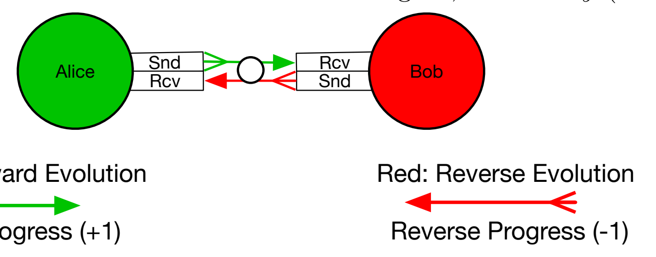
Many problems encountered in datacenters today arise from our inability to distinguish between a node that is merely slow from a node that has failed or become unreachable due to a network failure.

We take the two most recognizable elements in datacenters today: *servers* and *switches*, and refactor them into simpler, more foundational elements (fully *independent* failure domains): `cells` and `links`. A `cell` is a single type of node element (autonomous unit of compute, storage and packet processing). A `link` is an individual, bidirectional, computation object (an autonomous communication entity between *two* `cells`)[1].

A consequence of the former is that unifying node elements makes things simpler because we have only one type of node to manage instead of two. The consequence of the latter is profoundly more interesting: we raise the notion of a `link` to *first order* – a first-class citizen in the infrastructure – a bipartite[2] element of information with two *complementary* halves – *persistable* through failure and recovery events. i.e., a communication object that doesn’t rule out that some fault-detection and computation is involved.

An example `link` utility is *The I Know That You Know That I Know* (`TIKTYKTIK`) property; which enables us to address some of the most difficult and pernicious problems in *distributed systems* today.

Another example `link` utility is *Atomic Information Transfer* (`AIT`). Unlike *replicated* state machines[3] used throughout distributed applications today, `links` are *single* state machines: the two halves of which maintain *temporal intimacy* through hidden packet exchanges. When a local agent or actor is ready, the `AIT` protocol transfers *indivisible* tokens across the link to the other agent, *atomically* (all or nothing)[4].



These `TIKTYKTIK` and `AIT` properties are *composable*. Trees of `links` provide a resilient *conserved quantities* mechanism to reliably distribute tokens among *agents* on an application graph. Intermediate cells *promise*[5] to never lose `AIT` tokens. This defends against lost tokens because if any part of the chain (or tree) breaks, alternate paths are available to seamlessly recover the conserved quanity and continue operation.

By strengthening the system model, `links` and `AIT` provide a general foundation to solve many distributed systems problems[6], such as failure-detection, consensus and distributed transactions.

Failure Modes

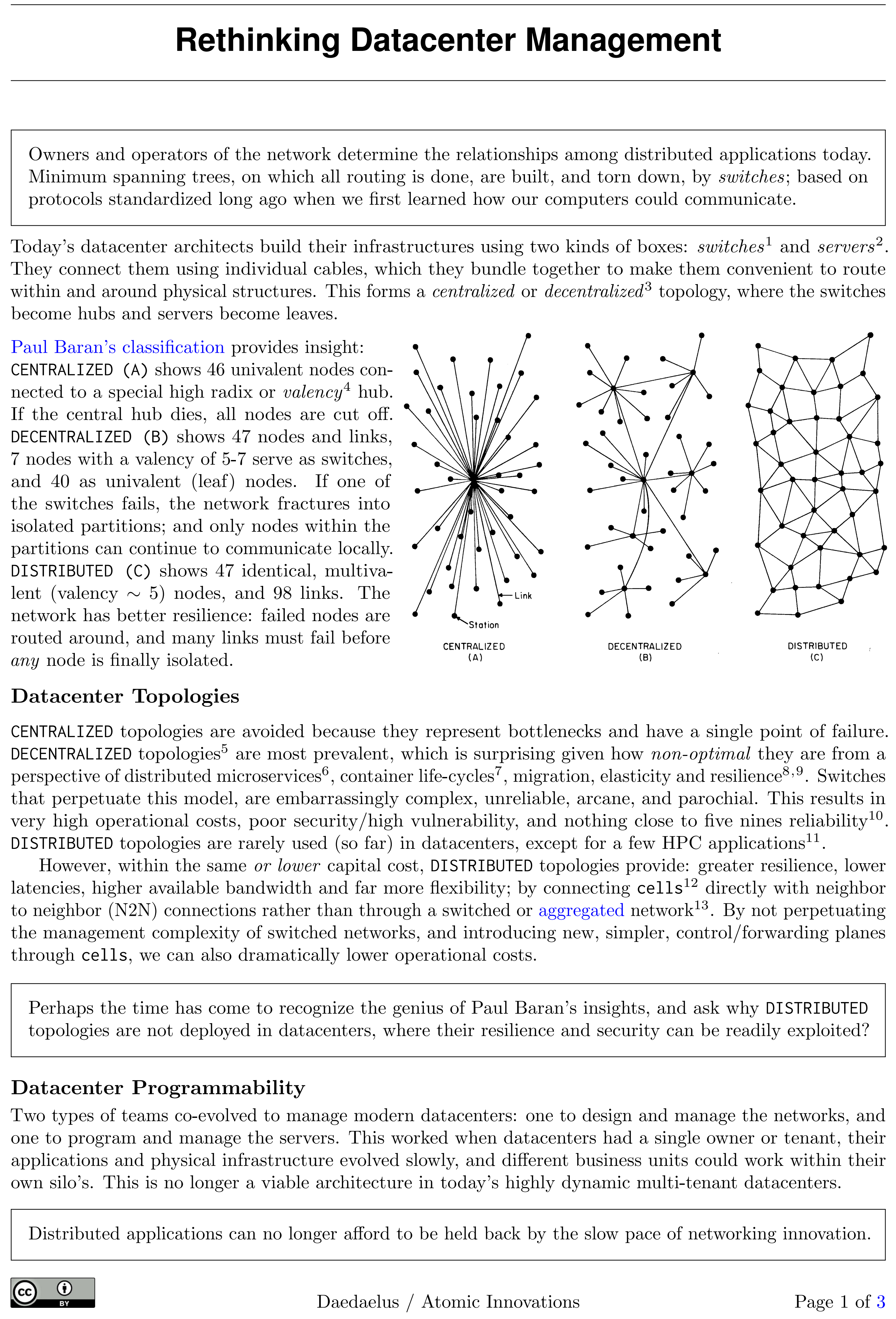
One might imagine we could achieve the properties of `links` over existing switched networks. If each host (or its NIC) maintains its half of the *shared state*, then shouldn’t the switched network be able to act as a proxy for a single *logical* link? When a switched network fails, and reroutes, can’t the two sides (NICs) just stitch the two halves of the *shared state* back together again?[7]

This simple hazard analysis[8] misses a fundamental issue: *networks don’t maintain state on behalf of applications*. Switches drop packets (and state) whenever *they* feel like it, so there are many more ways for *logical links* to get confused over switched networks and compromise the integrity of the *shared state*.

Key issue: Switched networks may drop packets anywhere along the path; eradicating state *and* events needed to maintain *promises* and *liveness* respectively. When a `link` fails, both sides are preserved. If there is a failure in the *token transfer* it can always be detected, and retransmissions occur only on a *real* failure (such as disconnection–where alternative routes are explicitly coordinated with applications), thus enforcing that tokens have no duplicate or out of order deliveries on the `link`[9,10].

Daedaelus / Atomic Innovations Page 1 of 3

Summary-Links.pdf

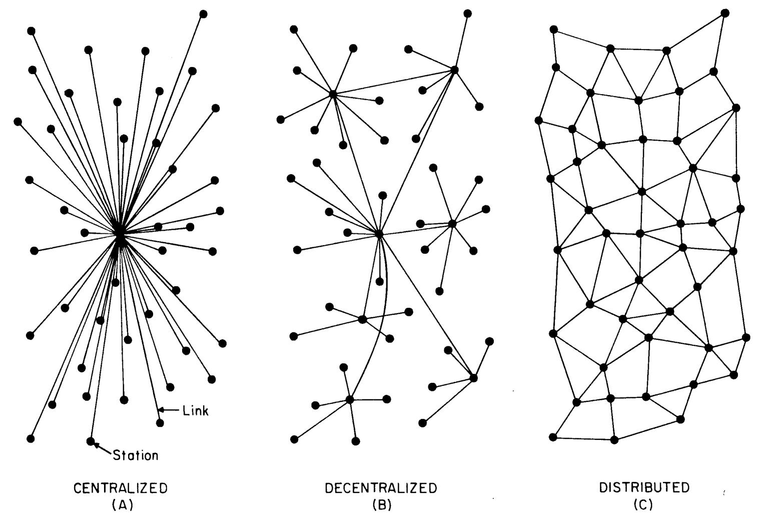


Rethinking Datacenter Management

Owners and operators of the network determine the relationships among distributed applications today. Minimum spanning trees, on which all routing is done, are built, and torn down, by *switches*; based on protocols standardized long ago when we first learned how our computers could communicate.

Today’s datacenter architects build their infrastructures using two kinds of boxes: *switches*[1] and *servers*[2]. They connect them using individual cables, which they bundle together to make them convenient to route within and around physical structures. This forms a *centralized* or *decentralized*[3] topology, where the switches become hubs and servers become leaves.

Paul Baran’s classification provides insight: `CENTRALIZED (A)` shows 46 univalent nodes connected to a special high radix or *valency*[4] hub. If the central hub dies, all nodes are cut off. `DECENTRALIZED (B)` shows 47 nodes and links, 7 nodes with a valency of 5-7 serve as switches, and 40 as univalent (leaf) nodes. If one of the switches fails, the network fractures into isolated partitions; and only nodes within the partitions can continue to communicate locally. `DISTRIBUTED (C)` shows 47 identical, multivalent (valency $\sim$ 5) nodes, and 98 links. The network has better resilience: failed nodes are routed around, and many links must fail before *any* node is finally isolated.



Datacenter Topologies

`CENTRALIZED` topologies are avoided because they represent bottlenecks and have a single point of failure. `DECENTRALIZED` topologies[5] are most prevalent, which is surprising given how *non-optimal* they are from a perspective of distributed microservices[6], container life-cycles[7], migration, elasticity and resilience[8,9]. Switches that perpetuate this model, are embarrassingly complex, unreliable, arcane, and parochial. This results in very high operational costs, poor security/high vulnerability, and nothing close to five nines reliability[10]. `DISTRIBUTED` topologies are rarely used (so far) in datacenters, except for a few HPC applications[11].

However, within the same *or lower* capital cost, `DISTRIBUTED` topologies provide: greater resilience, lower latencies, higher available bandwidth and far more flexibility; by connecting `cells`[12] directly with neighbor to neighbor (N2N) connections rather than through a switched or aggregated network[13]. By not perpetuating the management complexity of switched networks, and introducing new, simpler, control/forwarding planes through `cells`, we can also dramatically lower operational costs.

Perhaps the time has come to recognize the genius of Paul Baran’s insights, and ask why `DISTRIBUTED` topologies are not deployed in datacenters, where their resilience and security can be readily exploited?

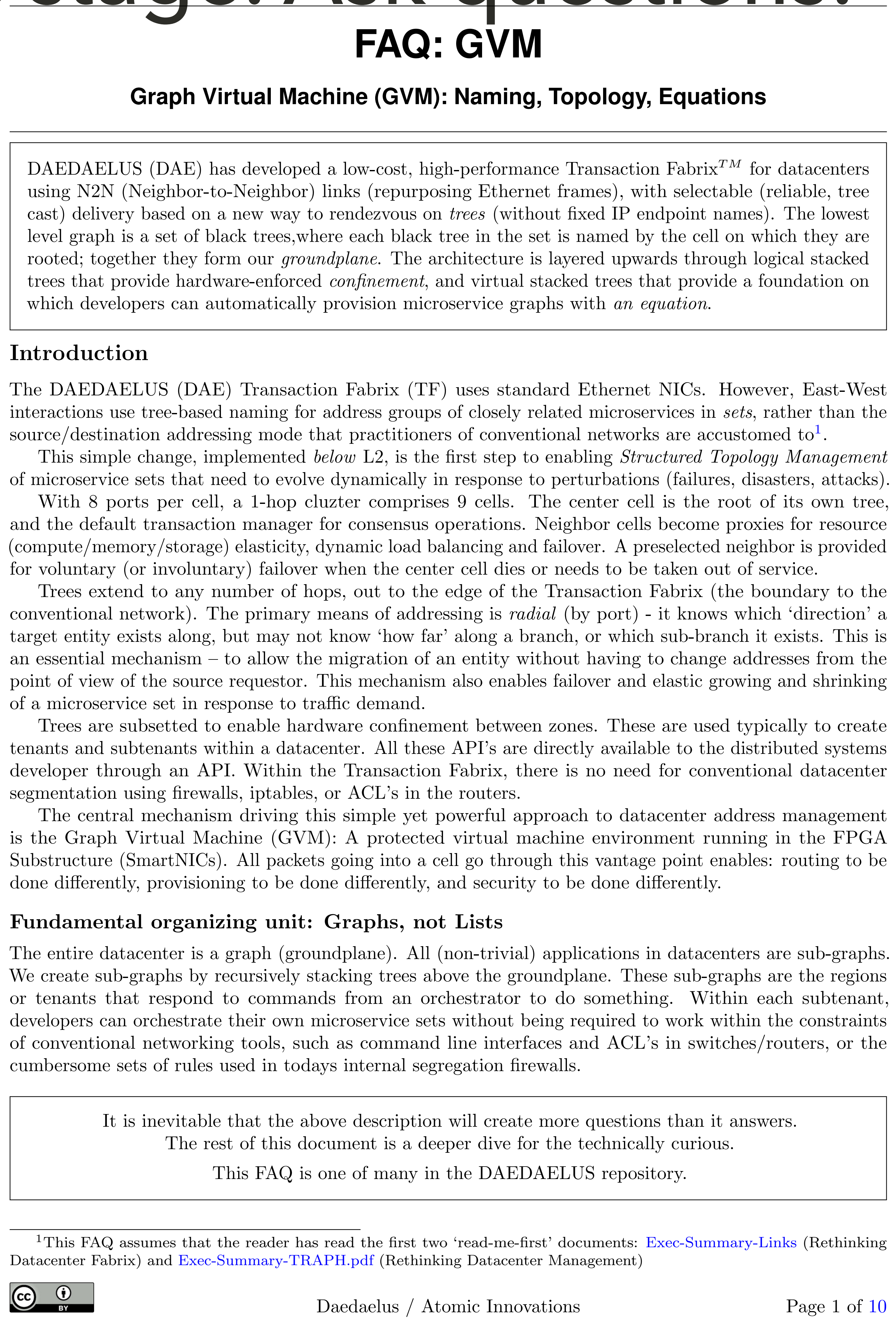
Datacenter Programmability

Two types of teams co-evolved to manage modern datacenters: one to design and manage the networks, and one to program and manage the servers. This worked when datacenters had a single owner or tenant, their applications and physical infrastructure evolved slowly, and different business units could work within their own silo’s. This is no longer a viable architecture in today’s highly dynamic multi-tenant datacenters.

Distributed applications can no longer afford to be held back by the slow pace of networking innovation.

Daedaelus / Atomic Innovations Page 1 of 3

Rethinking-Datacenter-Management.pdf



FAQ: GVM

Graph Virtual Machine (GVM): Naming, Topology, Equations

DAEDAELUS (DAE) has developed a low-cost, high-performance Transaction Fabrix^{TM} for datacenters using N2N (Neighbor-to-Neighbor) links (repurposing Ethernet frames), with selectable (reliable, tree cast) delivery based on a new way to rendezvous on *trees* (without fixed IP endpoint names). The lowest level graph is a set of black trees,where each black tree in the set is named by the cell on which they are rooted; together they form our *groundplane*. The architecture is layered upwards through logical stacked trees that provide hardware-enforced *confinement*, and virtual stacked trees that provide a foundation on which developers can automatically provision microservice graphs with *an equation*.

Introduction

The DAEDAELUS (DAE) Transaction Fabrix (TF) uses standard Ethernet NICs. However, East-West interactions use tree-based naming for address groups of closely related microservices in *sets*, rather than the source/destination addressing mode that practitioners of conventional networks are accustomed to[1].

This simple change, implemented *below* L2, is the first step to enabling *Structured Topology Management* of microservice sets that need to evolve dynamically in response to perturbations (failures, disasters, attacks).

With 8 ports per cell, a 1-hop cluzter comprises 9 cells. The center cell is the root of its own tree, and the default transaction manager for consensus operations. Neighbor cells become proxies for resource (compute/memory/storage) elasticity, dynamic load balancing and failover. A preselected neighbor is provided for voluntary (or involuntary) failover when the center cell dies or needs to be taken out of service.

Trees extend to any number of hops, out to the edge of the Transaction Fabrix (the boundary to the conventional network). The primary means of addressing is *radial* (by port) - it knows which ‘direction’ a target entity exists along, but may not know ‘how far’ along a branch, or which sub-branch it exists. This is an essential mechanism – to allow the migration of an entity without having to change addresses from the point of view of the source requestor. This mechanism also enables failover and elastic growing and shrinking of a microservice set in response to traffic demand.

Trees are subsetted to enable hardware confinement between zones. These are used typically to create tenants and subtenants within a datacenter. All these API’s are directly available to the distributed systems developer through an API. Within the Transaction Fabrix, there is no need for conventional datacenter segmentation using firewalls, iptables, or ACL’s in the routers.

The central mechanism driving this simple yet powerful approach to datacenter address management is the Graph Virtual Machine (GVM): A protected virtual machine environment running in the FPGA Substructure (SmartNICs). All packets going into a cell go through this vantage point enables: routing to be done differently, provisioning to be done differently, and security to be done differently.

Fundamental organizing unit: Graphs, not Lists

The entire datacenter is a graph (groundplane). All (non-trivial) applications in datacenters are sub-graphs. We create sub-graphs by recursively stacking trees above the groundplane. These sub-graphs are the regions or tenants that respond to commands from an orchestrator to do something. Within each subtenant, developers can orchestrate their own microservice sets without being required to work within the constraints of conventional networking tools, such as command line interfaces and ACL’s in switches/routers, or the cumbersome sets of rules used in todays internal segregation firewalls.

It is inevitable that the above description will create more questions than it answers.
The rest of this document is a deeper dive for the technically curious.

This FAQ is one of many in the DAEDAELUS repository.

[1]This FAQ assumes that the reader has read the first two ‘read-me-first’ documents: Exec-Summary-Links (Rethinking Datacenter Fabrix) and Exec-Summary-TRAPH.pdf (Rethinking Datacenter Management)

Daedaelus / Atomic Innovations Page 1 of 10

FAQ-GVM.pdf

DÆDÆLUS

Team Responsibilities

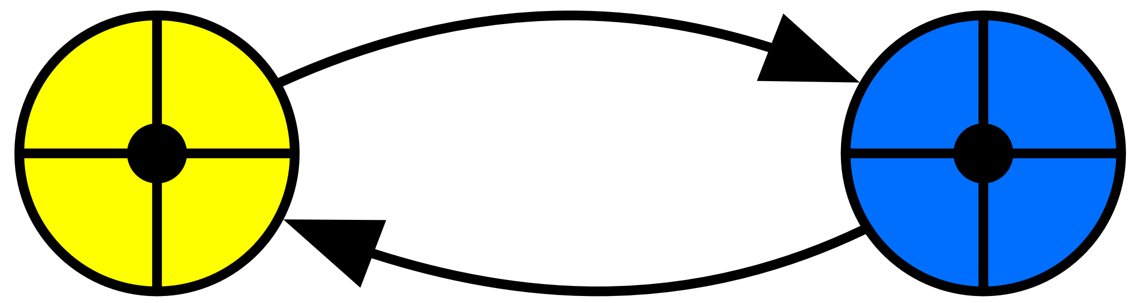
- [Paul] Define vision, mission, use cases, product requirements
- [Steve] Technology Context: Networking, Compute, Storage
- [Sahas] Emulation Rig: Transaction Protocol, Python Instrumentation
- [Dugan] Simulation, Visualization, Artistry
- [Kevin] Detector Rig for Time Synchronization Anomalies
- [David?] Bits on the Wire Tools. RTL/Verilog, Measurement?
- [Others?] – there's lots of work to be done –

Multiway Link Protocol

Trapped (circulating) "information" token on link, exchanged with strata above, or strata below for token *conservation* in and between layers

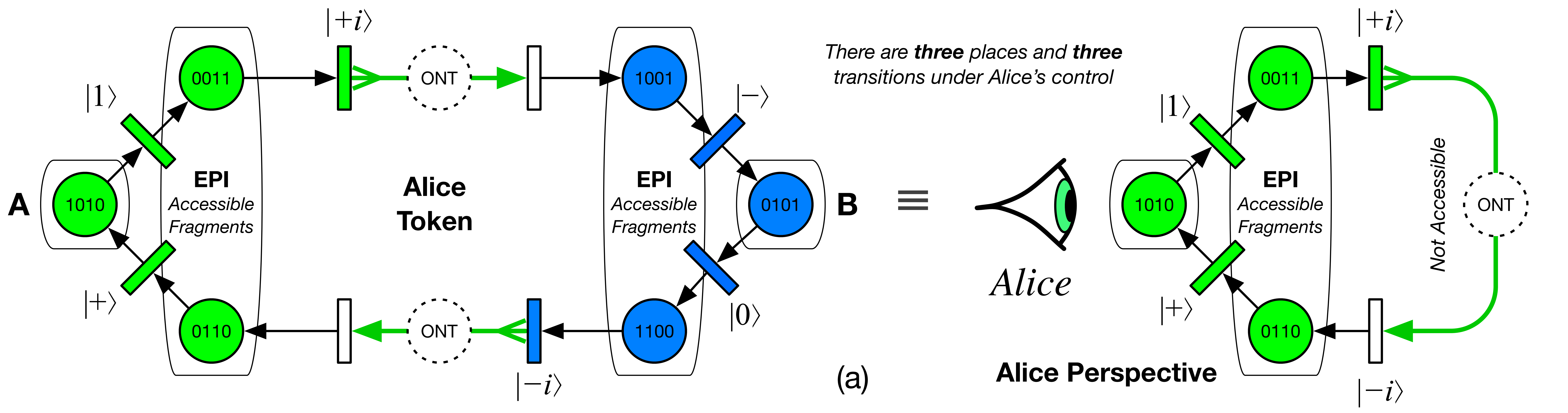
- Interfaces for logical time (Lingua Franca) reversal ancilla for reactors
- Causal invariance leads to precise definition of confined Shannon information recovery (based on knowledge from Cluster Liveness), in the face of *all hazards* presented via fault injection, via externally accessible ancilla from the *next* level up. i.e., the ONT Petri place and oracle

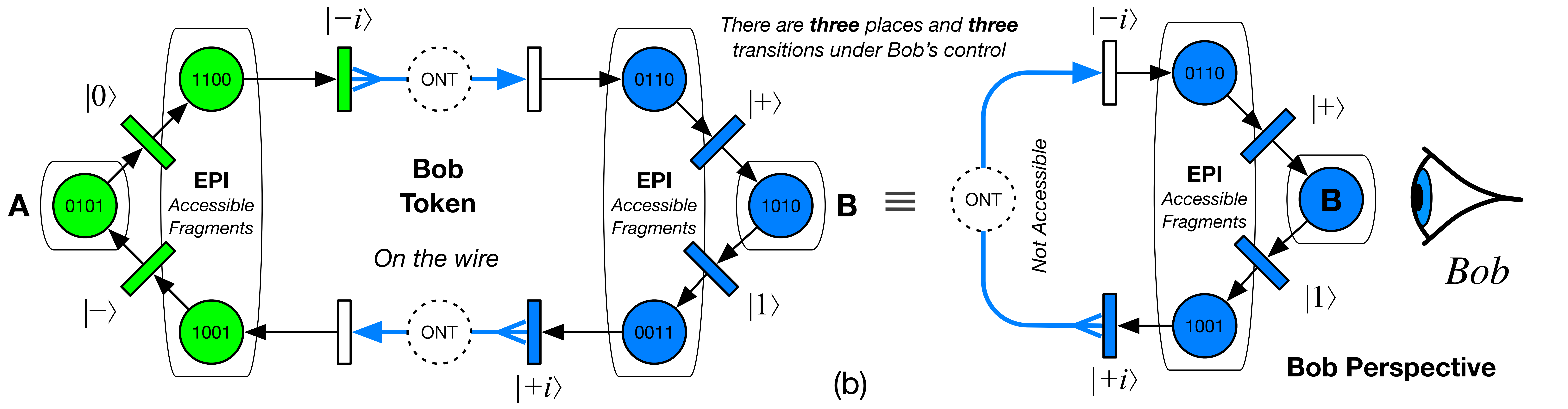
Superposition of Arrow Head & Tail.
Indicates Alternating causality is
occurring under the hood, but you can't see
It because reads are destructive.



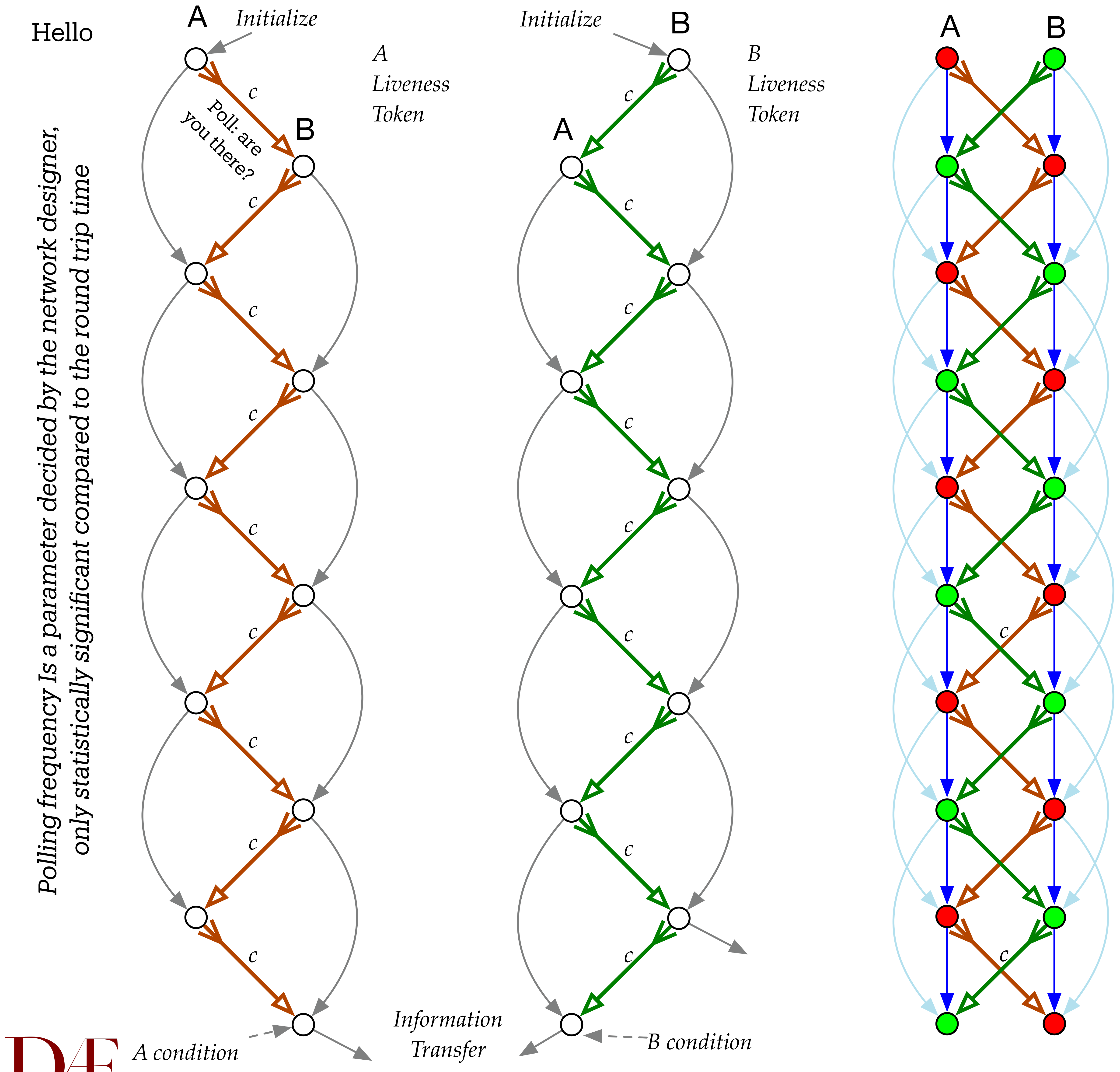
We separate the physics of entanglement
From the mathematics of entanglement.

Dual SAW Petri Spekkens





Ethernet Link Liveness


Hello
A
Initialize
B
Poll: are you there?
c
A Liveness Token
B Liveness Token
Polling frequency Is a parameter decided by the network designer, only statistically significant compared to the round trip time
Information Transfer
A condition
B condition

Alternating Causality protocol

Imagine an Ethernet cable connecting two SmartNICs

Liveness provides symmetric question & answer dialog:

Are you there? ⇆ Yes I am! (First slice of frame)

Each frame may be modified *only by the receiver* when it has the frame in its possession. There is only one serializability focus per frame, but there are two frames: one owned by Alice, one owned by Bob.

- In Alice's frame we say that Alice's token is owned, but Bob's token is borrowed.
- In Bob's frame we say that Bob's token is owned, but Alice's token is borrowed.

Following Rust rules.

MultiWay View of Link

An example of a dynamic yet stable causal structure (multicomputation reducibility) using a dual token Petri net, spanning a single bipartite link. Quantum Ethernet. Dual SAW-Petri-Spekkens Protocol.

Imagine an Ethernet cable, with each end connected to a SmartNIC -- each with a server attached to the other side of it's PCIe or CXL bus. Conventional heartbeats enable \emph{liveness} on the link with a symmetric question and answer dialog: Are you there? <-> Yes I am!

The link continuously circulates fixed frames of shared information. The figure to the right shows **two** independently circulating shared frames green \& red

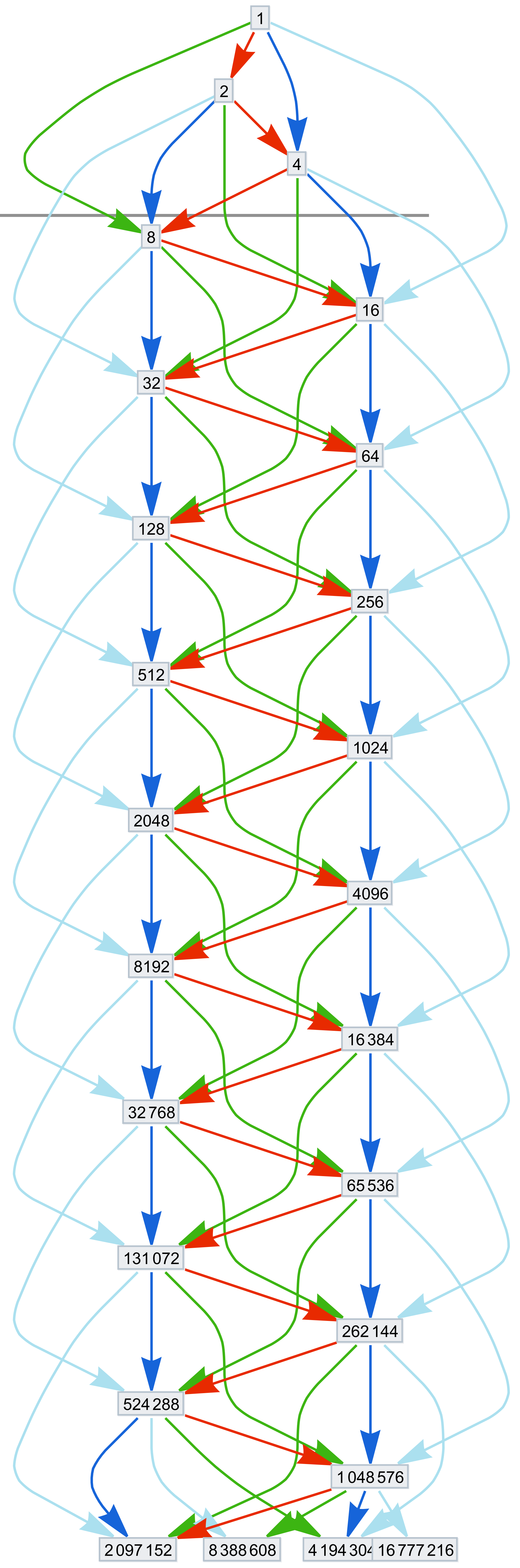
)\footnote{These two frames are shown as concurrent in this graph, but they will serialize, \\ \emph{convoying}: with `alternating causality' on a pair of Ethernet transmit/receive channels.}. Each frame may be modified only by the receiver when it has the frame in its possession. There is only one \emph{serializability} focus per frame, but there are two frames: one owned by Alice, one owned by Bob.

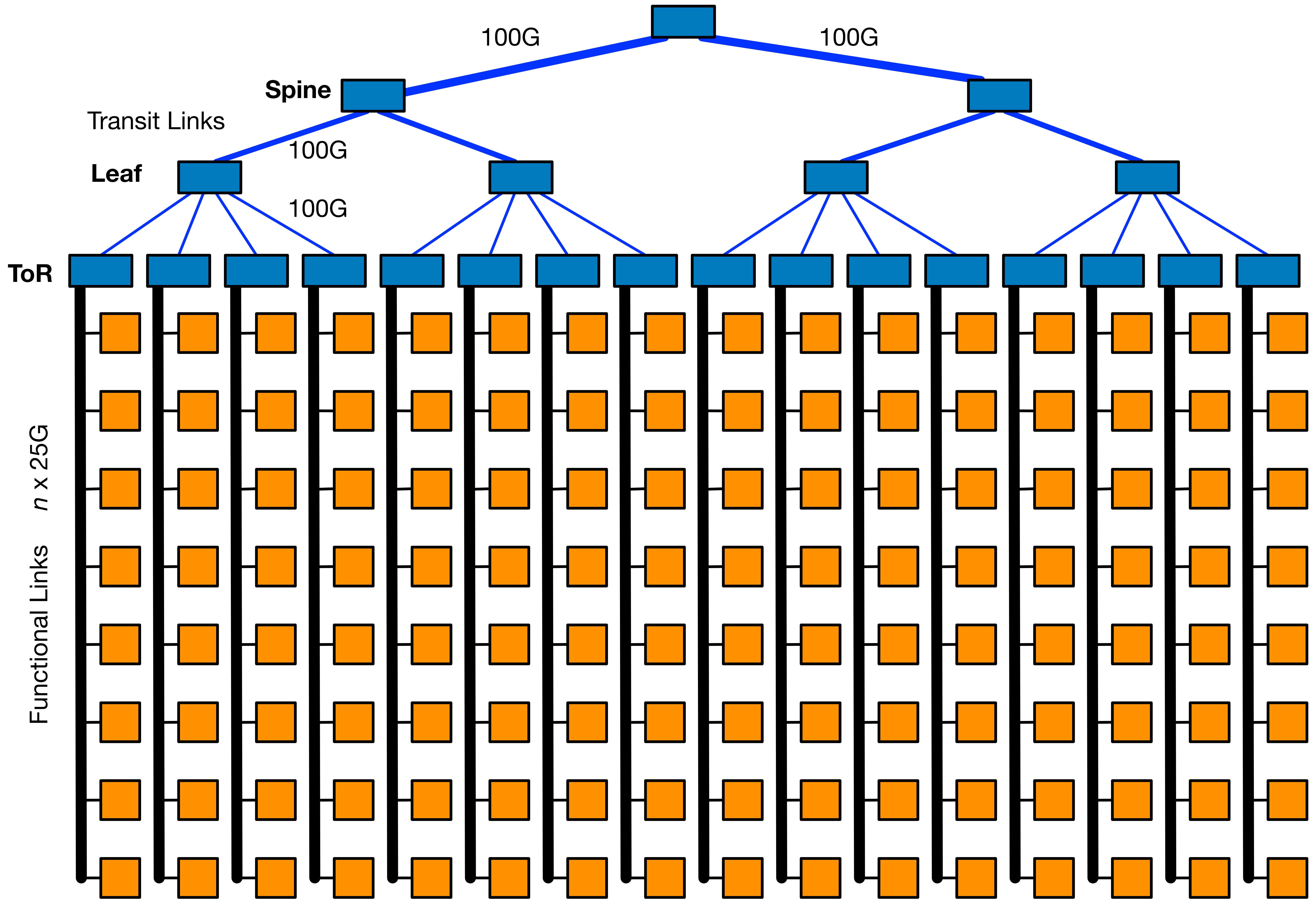
There is thus one circulating (direction) of causality.

Each causal frame has an identity (Alice or Bob) designated in the protocol after an appropriate symmetry breaking. In Alice's frame we say that Alice's token is \emph{owned,} but Bob's token is \emph{borrowed}. In Bob's frame we say that Bob's token is \emph{owned,} but Alice's token is \emph{borrowed}.

\subsection{Alice/Bob Independently initialized with a Petri Token}

\paragraph{[1] Begin with Alice} Alice sends two pieces of information: Red to the other side of the link, Blue to self (to merge with future operations).



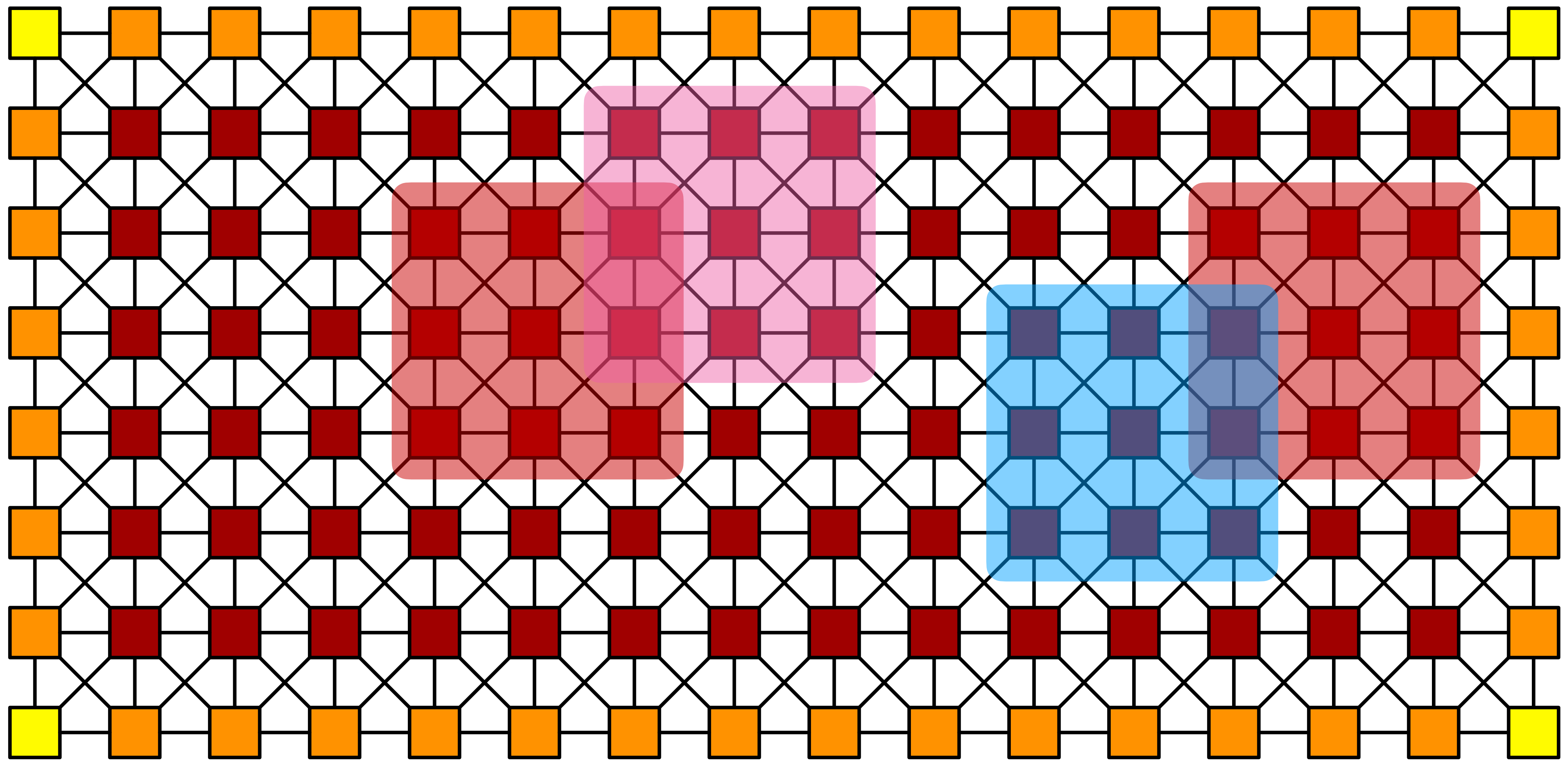


Conventional Clos

Graph (Network) Automata Foundations

8-port (Moore) neighborhood exploits physical link topology substructure within racks. But with *fragile* cabling (where links and nodes must be healed around dynamically with Local Observer View (LOV) information only)

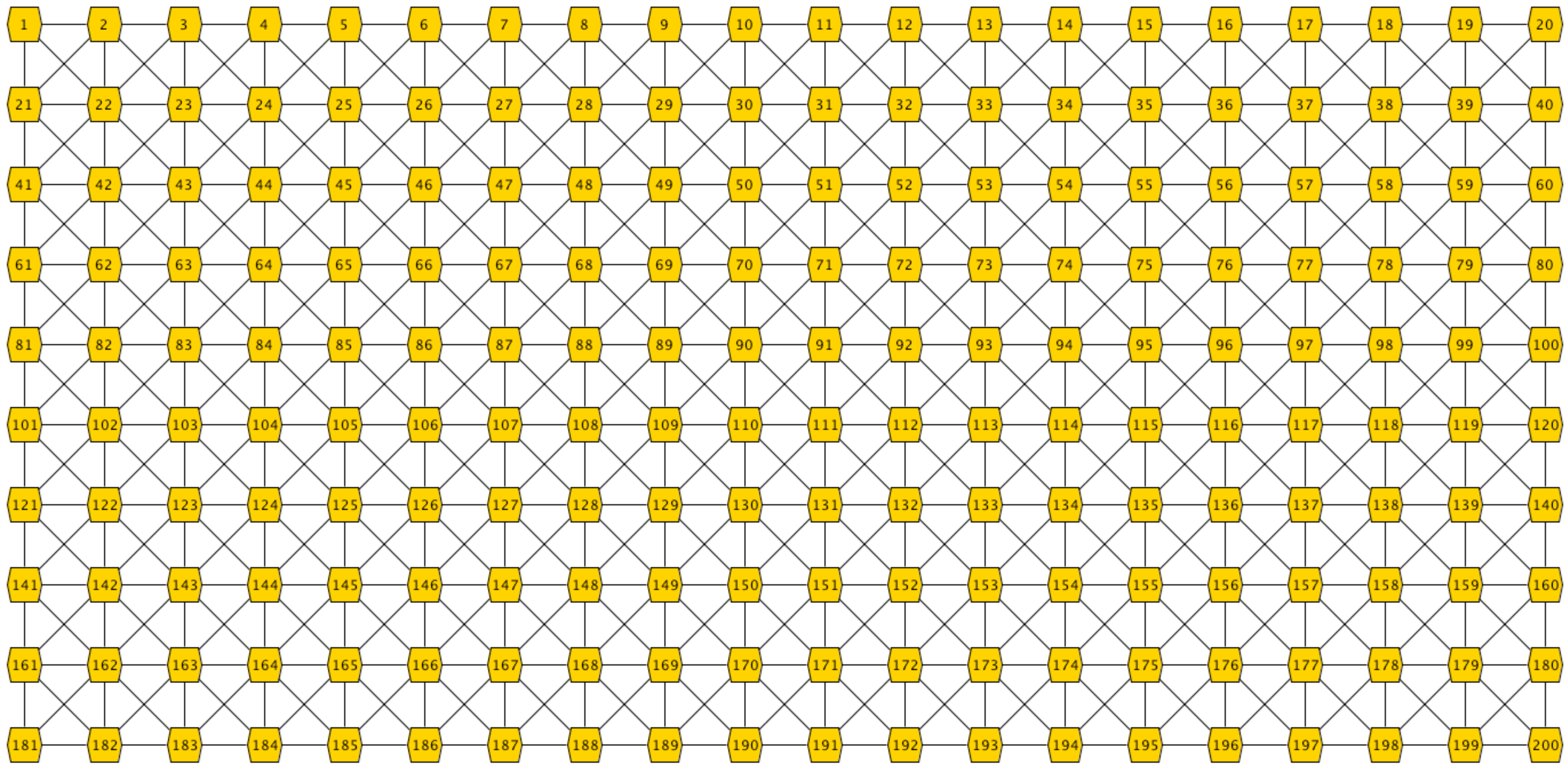
- Paves the way for highly dynamic computational strata in layers above
- Leads to a 9-Cell deployable unit of distributed computation we call a hypercell (Tile)*



*This physical structuring must not be conflated with a regular lattice on a cartesian coordinate system because our graph algorithms are highly dynamic, adapting to both the addition of new resources, or the deletion (i.e. failure or removal) of old ones. A fixed cartesian coordinate system (hypercube-like or otherwise) would be far too fragile as all labels throughout the graph would have to be replaced on failures and reconfigurations

Regular or Irregular Lattice?

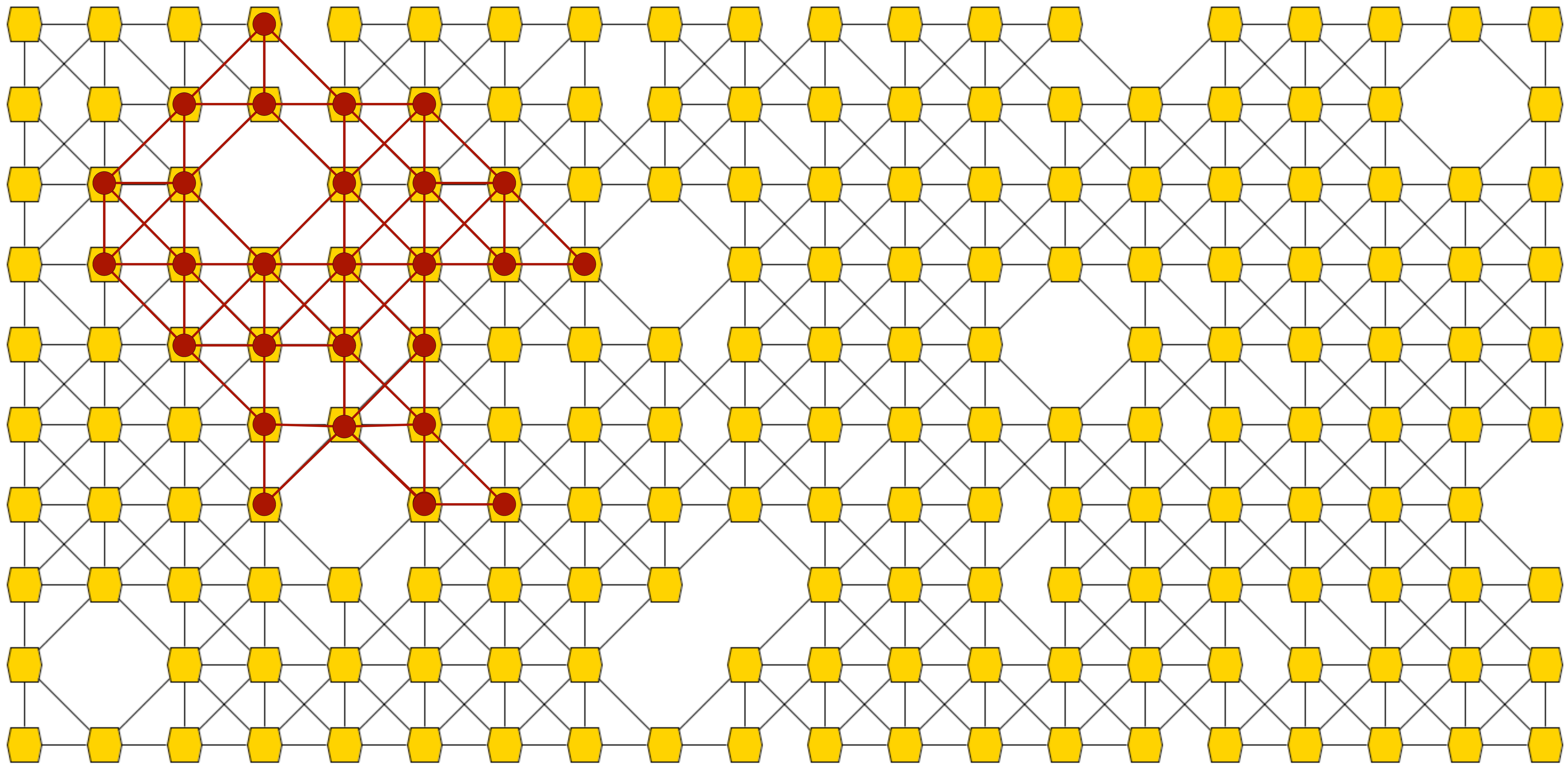
Real Infrastructures are messy



~~*Absolute / Cartesian Coordinate Addressing*~~

Neighbor to Neighbor (N2N) Lattice

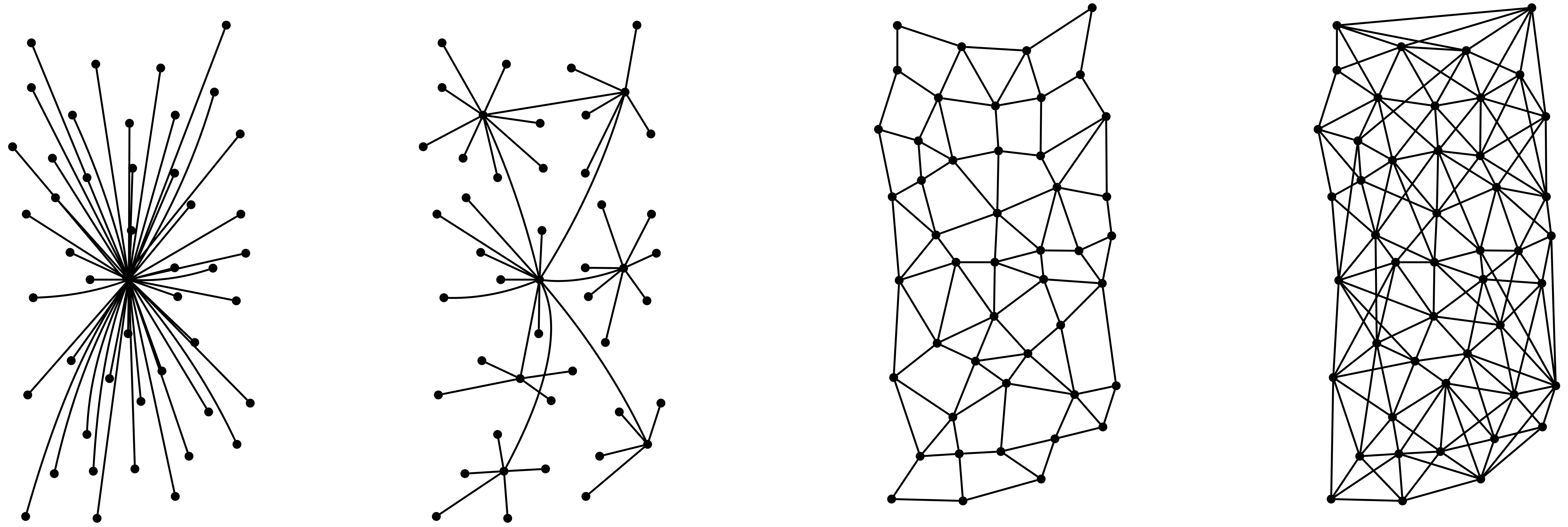
Manage on a Tree *Compute on a Graph*



Relative Address-Free Routing

Graph Virtual Machine (GVM)

Build and tear down 'named' graph relationships (sets/graph covers) based on properties of a cell or hypercell. Demonstrate using Paul Baran examples*:



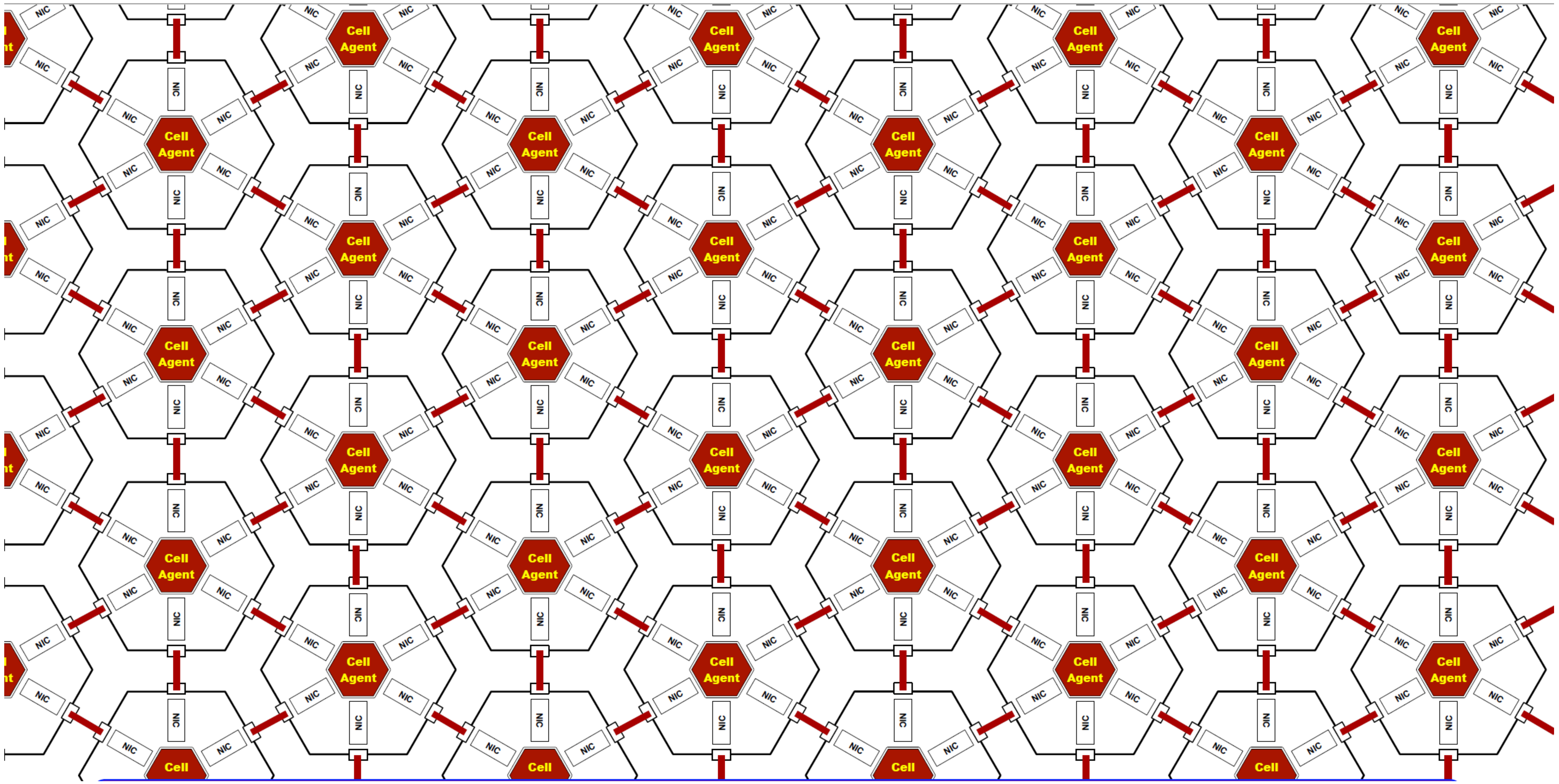
Count connectivity using: Matrix Tree Theorem & Graph Laplacian

Centralized	**Decentralized**	**Distributed**	**Transaction Fabrix**
(47 Nodes, 46 Links)	(47 Nodes, 47 Links)	(47 Nodes, 98 Links)	(47 Nodes, 220 Links)

*From : [Paul Baran, on Distributed communications.]

Logical Plane (mesh) of servers

Not a causal network yet (those are stacked on top)

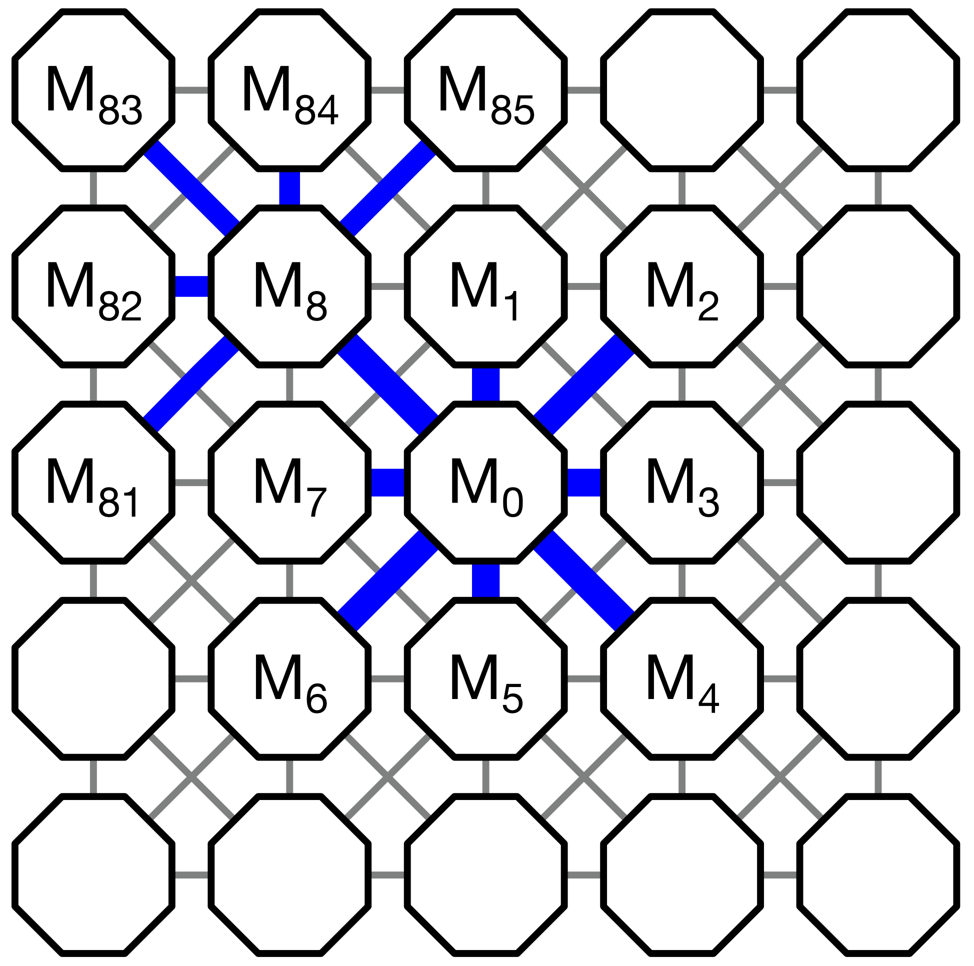


Composable (Stacked) TRAPHs

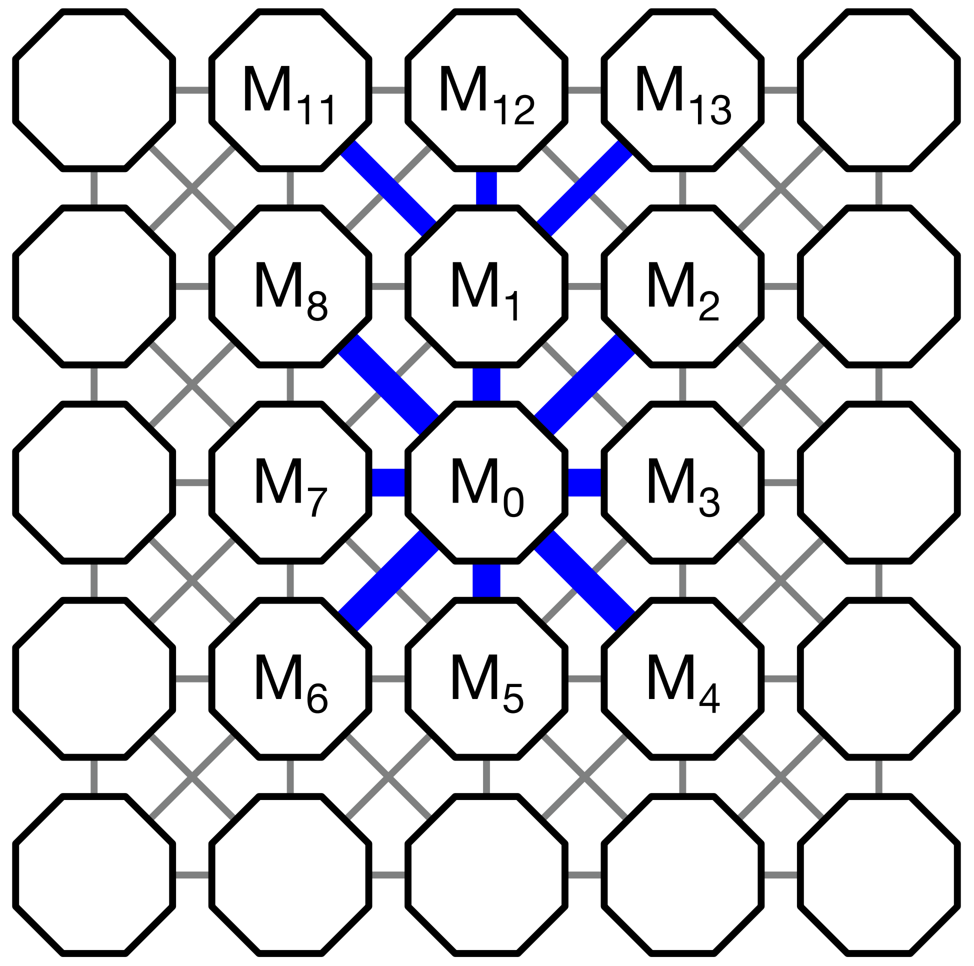
- Composable (Stacked) TRAPHs (TRee-grAPHs) Select one, or a small number of strata for experiment and visualization
- Use Cluster Liveness model to demonstrate spatial confinement
- Visualize TRAPHs for stacked trees, show physical, logical, virtual
- Demonstrate (local) distributed consensus within the tile. This 9-cell Tile also gives us a natural unit of fault-tolerance. Each cell is (by definition) the center of it's own tile
- The self-cell is thus pre-allocated in various algorithms as the center (Captain) of its own world for the purpose of initiating distributed algorithms, such as routing, consensus, autoscaling and load balancing

Hypercell Routing protocol

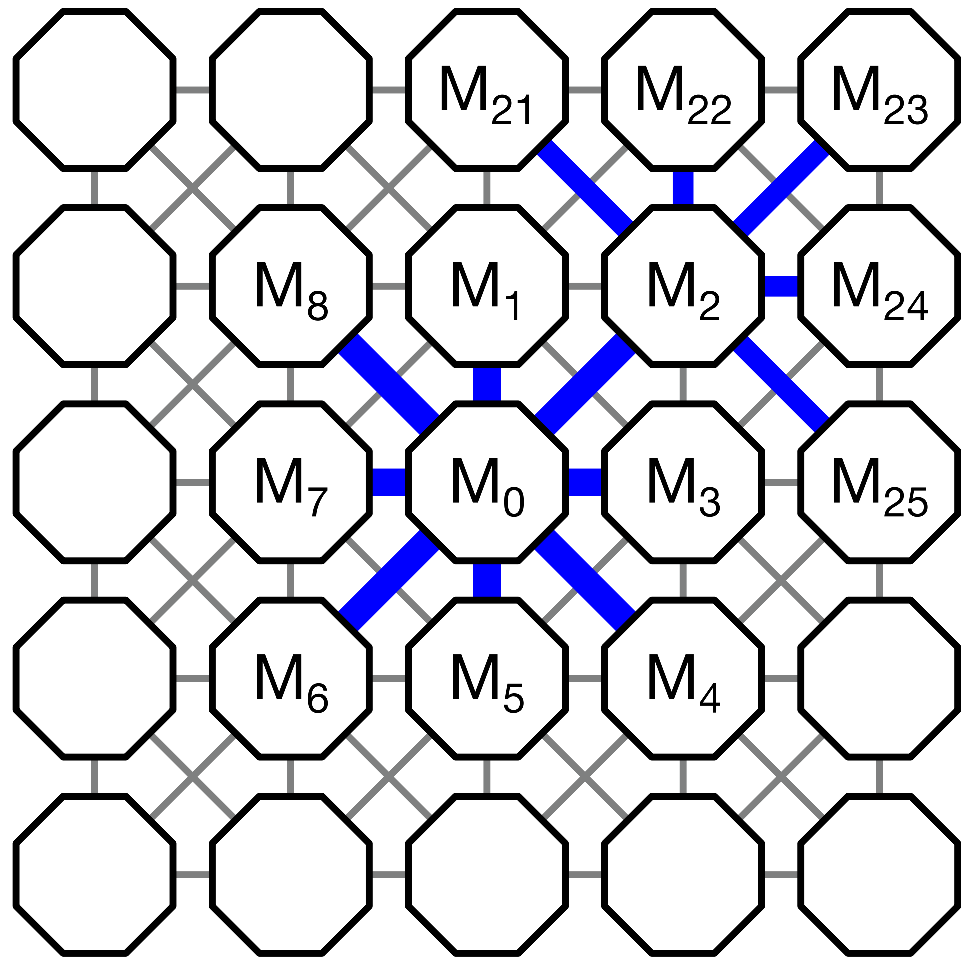
- Demonstrate Multiway routing protocol between tiles on DAGs
- Demonstrate healing around individual link failures
- Demonstrate remote consensus between tiles distributed throughout the graph, in the presence of injected failures



Tree Addressing from M8



Tree Addressing from M1

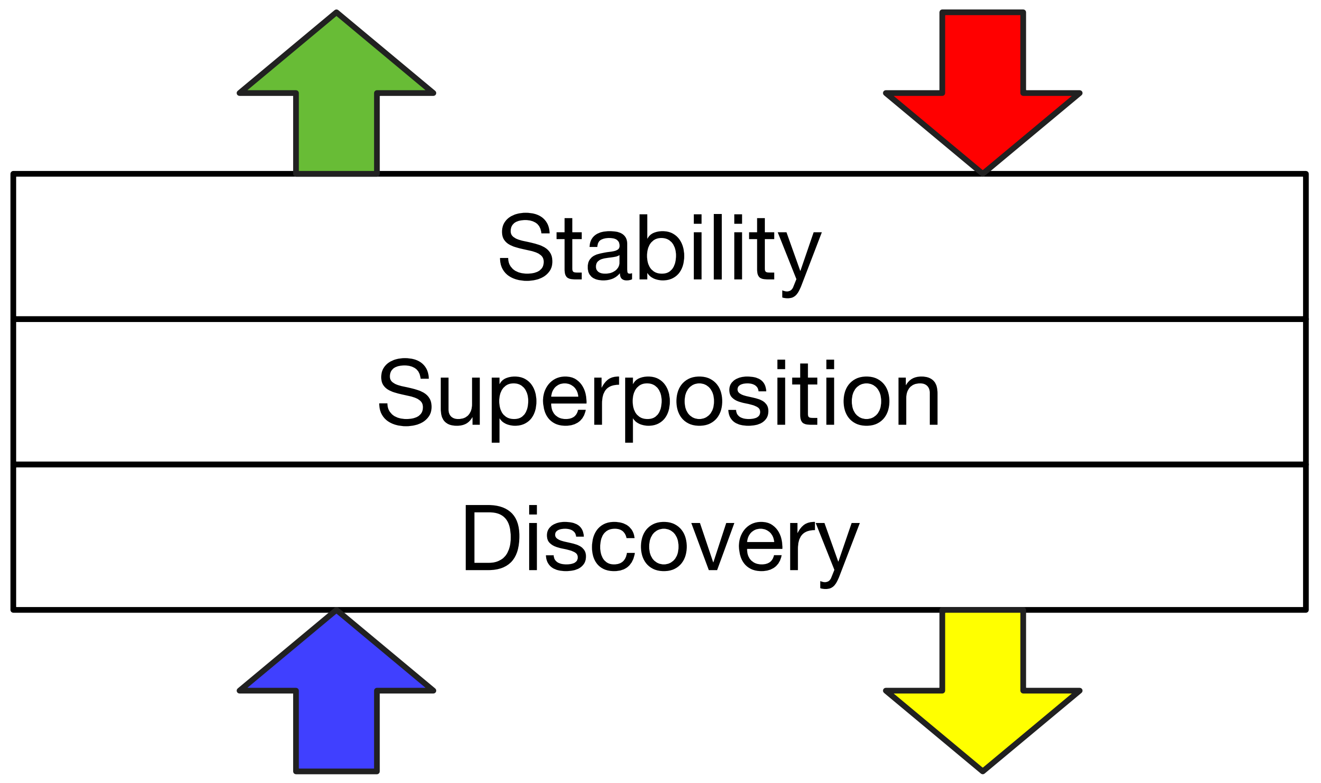


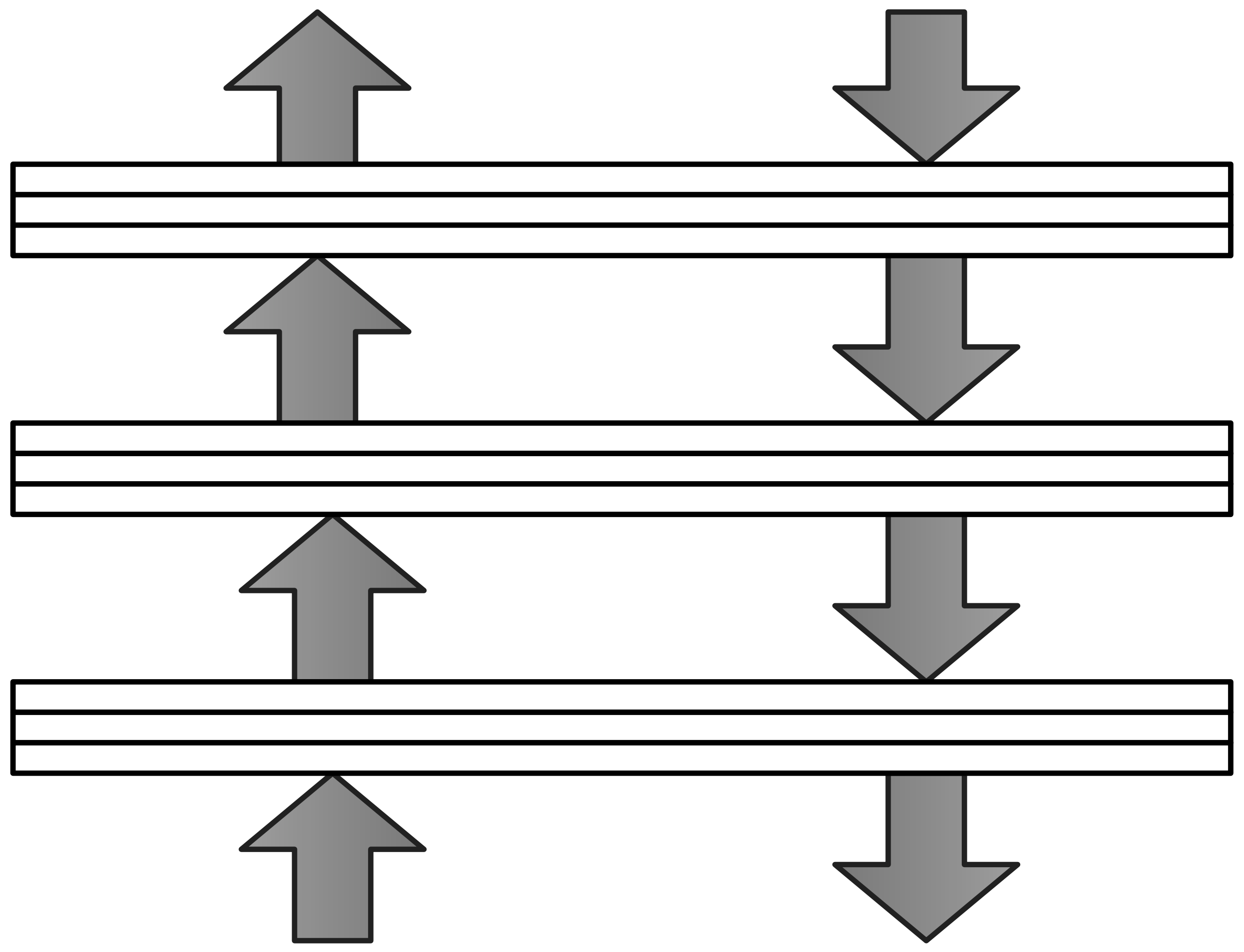
Tree Addressing from M2

Tree Addressing is *Relative* — It looks different from each Tree's perspective

Graph Stratification

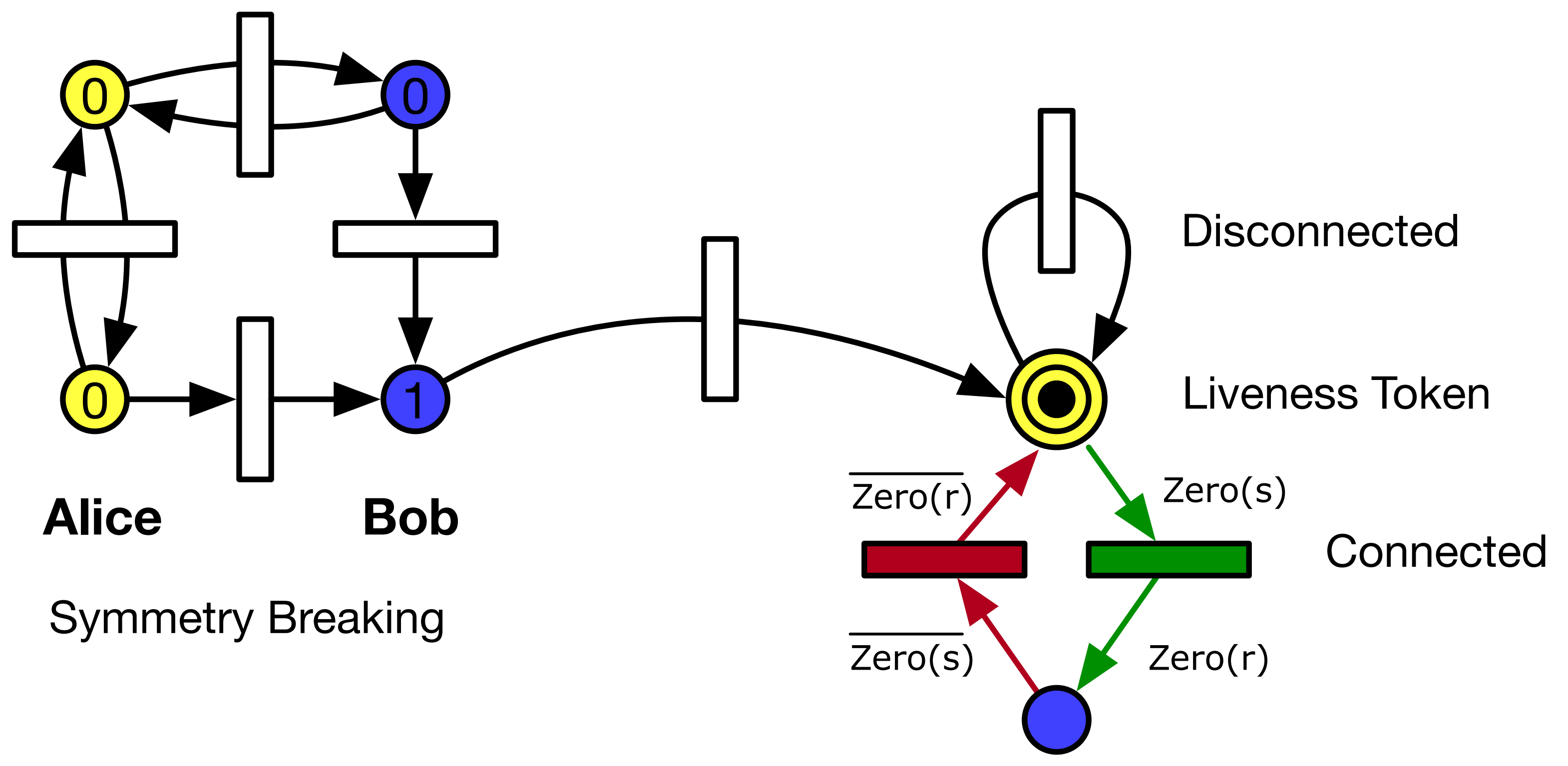
- Demonstrate coarse graining where hypergraphs represent multiple physical nodes 3 × 3 tiles which represents cooperative computational and failover entities
- 'Zoom out' to whole datacenter (all nodes), or 'Zoom in' to smaller sets of nodes in hypergraph style consolidation based on the same set of physical vertices and physical edges. Hypergraph can also 'go down' into the physical nodes and connect with sets of execution entities in a potentially hierarchical structure such as virtual machines, containers and lambdas
- Demonstrate 'Kubernetes replacement' for configuration and deployment





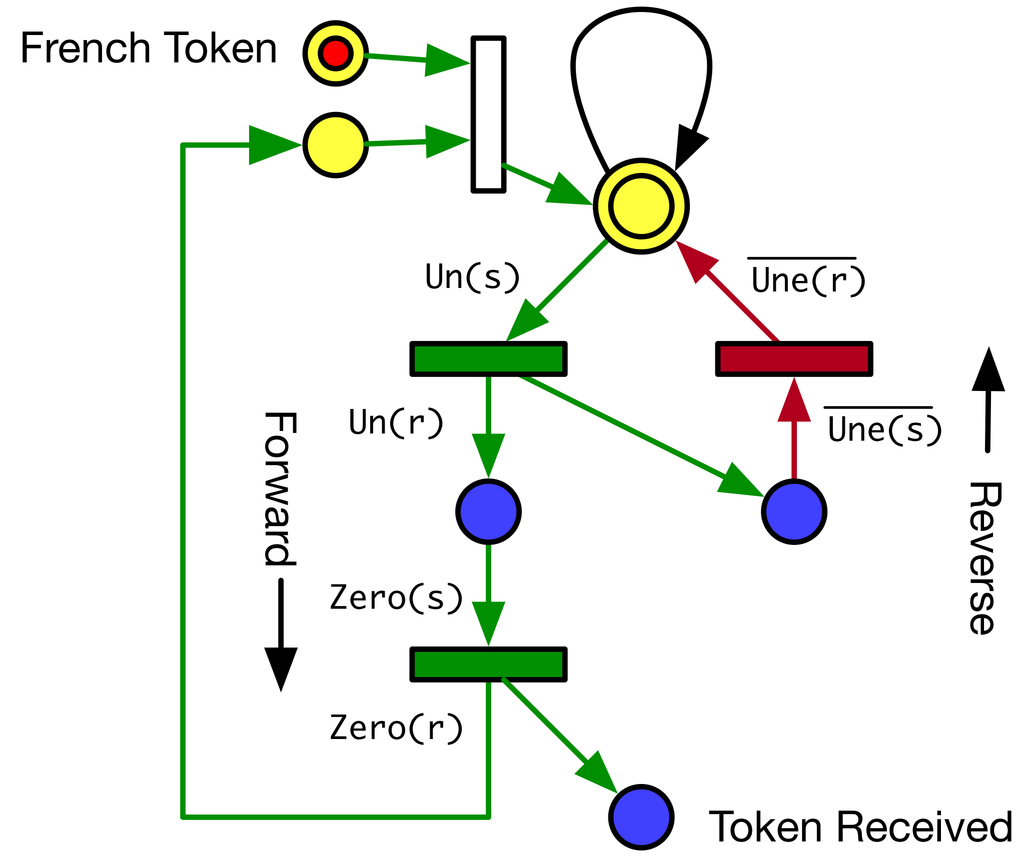
Token Dynamics

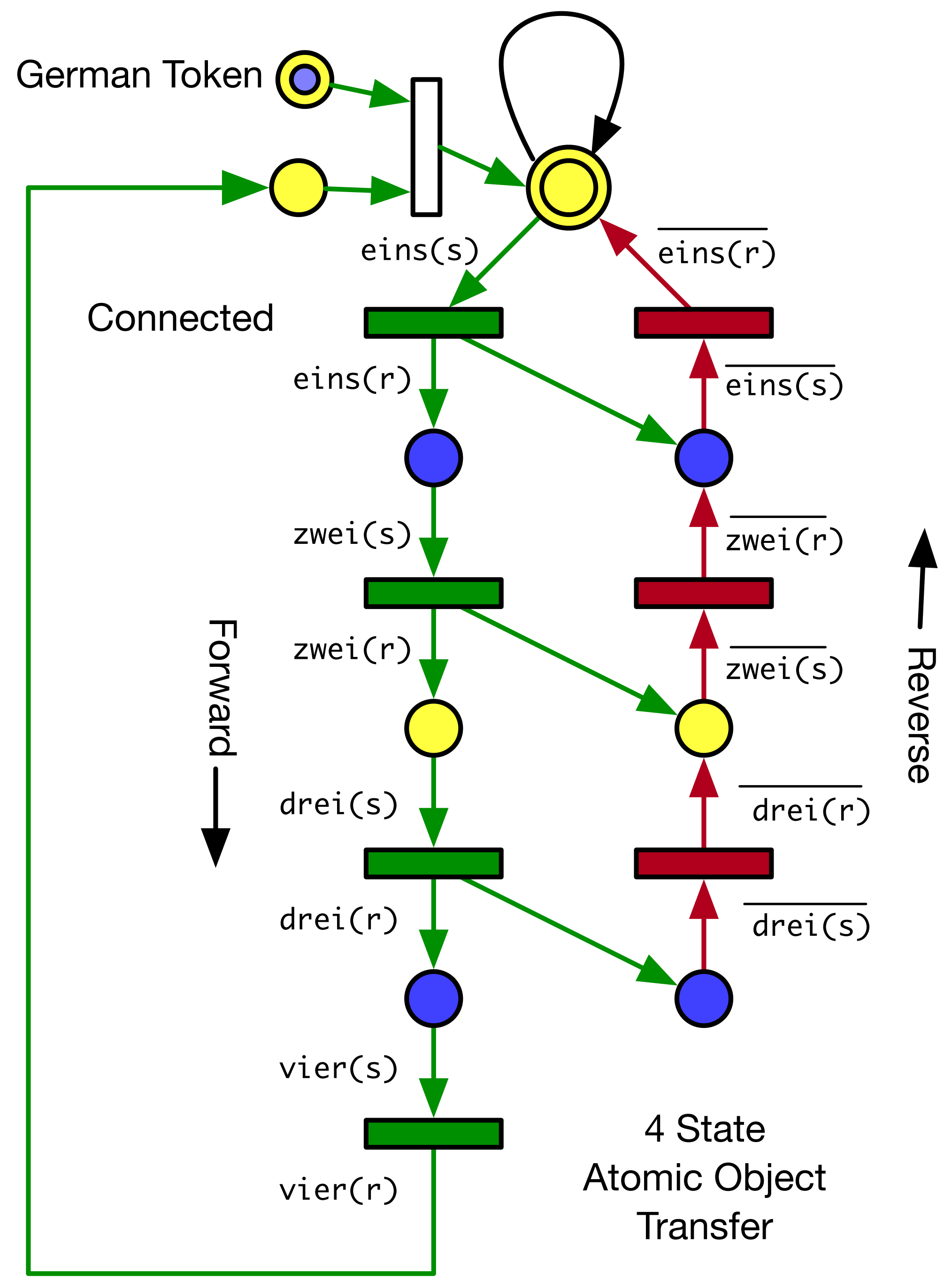
Conserved quantities/exchanged quantities/reversal recovery/lost baggage on *TRAPHs*. Include measurements such as graph resiliency, total counts for each token color. Include consistency checks to be tested after fault-injection of each kind of hazard (magnitude and phase errors in Spekkens states)



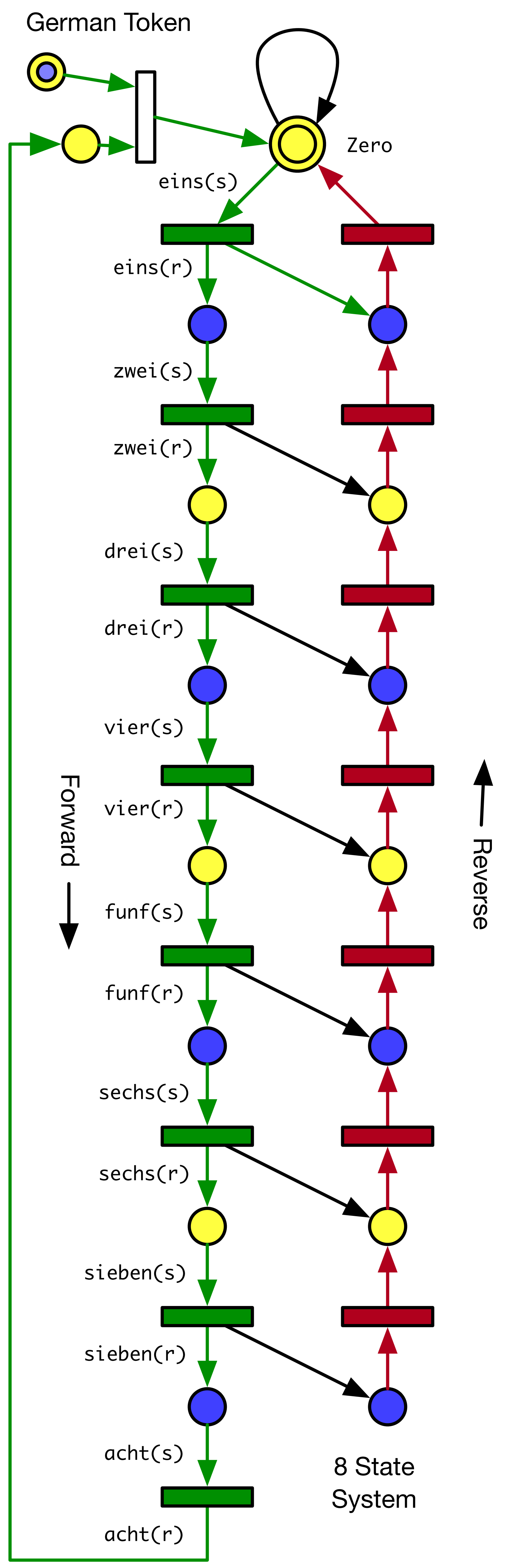
Symmetry Breaking

1 State Liveness Token



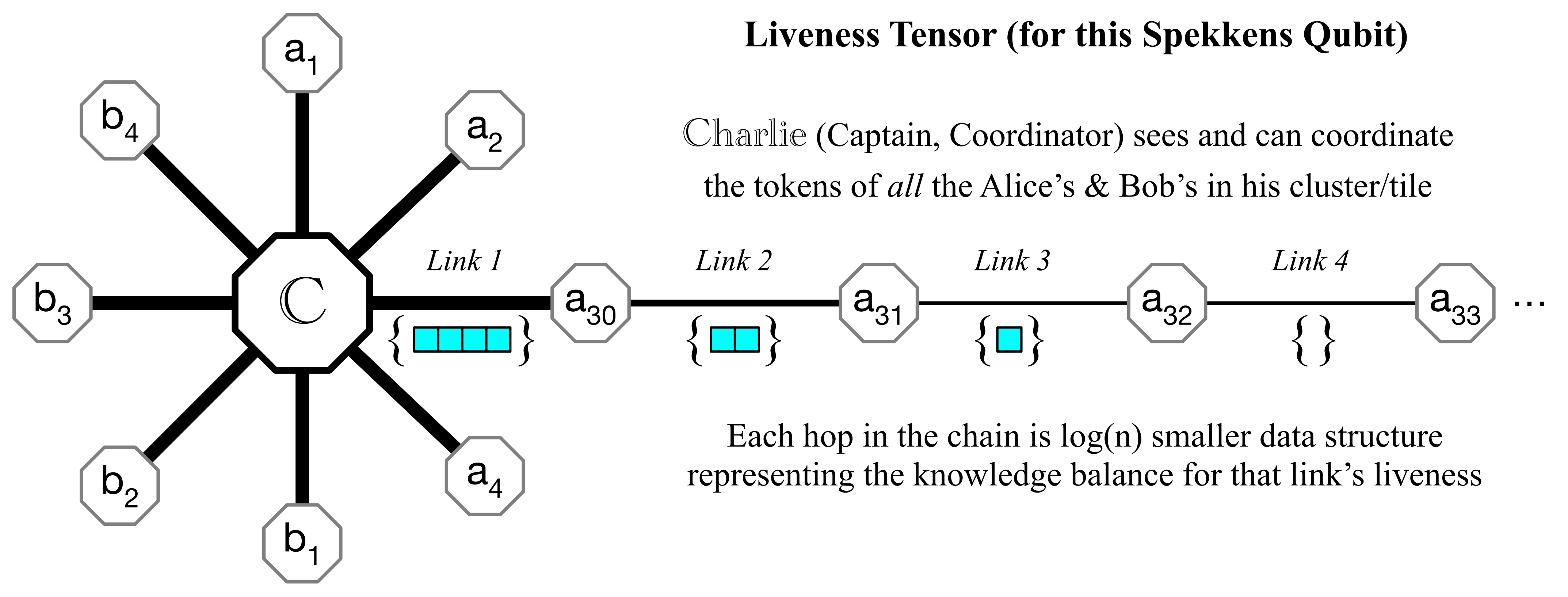

German Token
Connected
eins(s)
$\overline{\text{eins(r)}}$
eins(r)
$\overline{\text{eins(s)}}$
zwei(s)
$\overline{\text{zwei(r)}}$
zwei(r)
$\overline{\text{zwei(s)}}$
Forward
Reverse
drei(s)
$\overline{\text{drei(r)}}$
drei(r)
$\overline{\text{drei(s)}}$
vier(s)
vier(r)

4 State Atomic Object Transfer

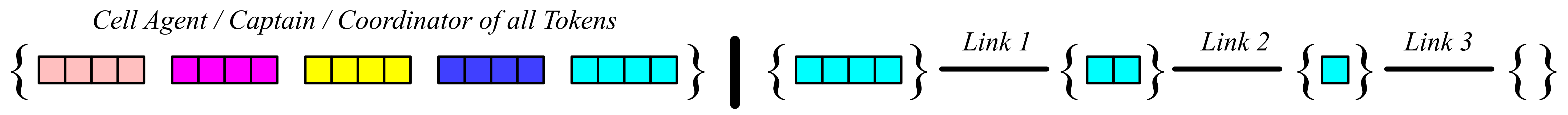


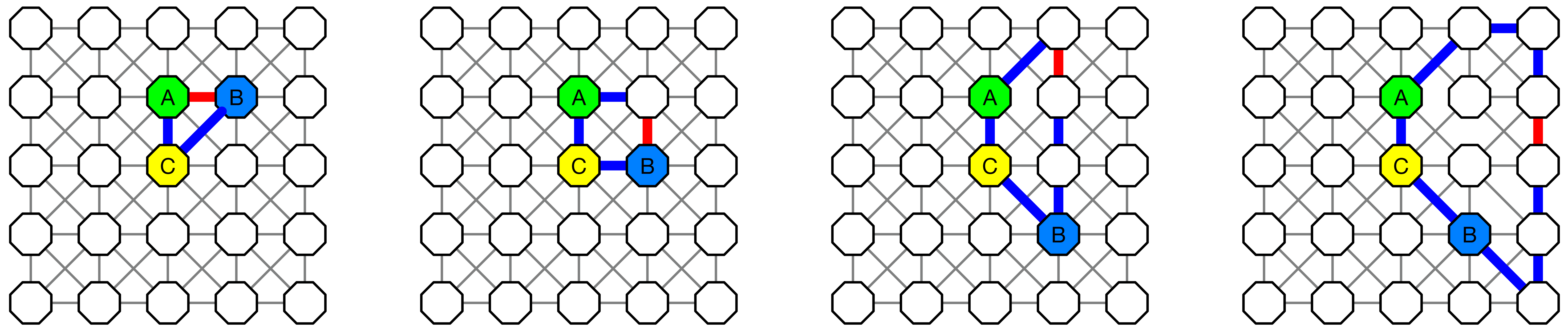
8 State System

Liveness Tensor

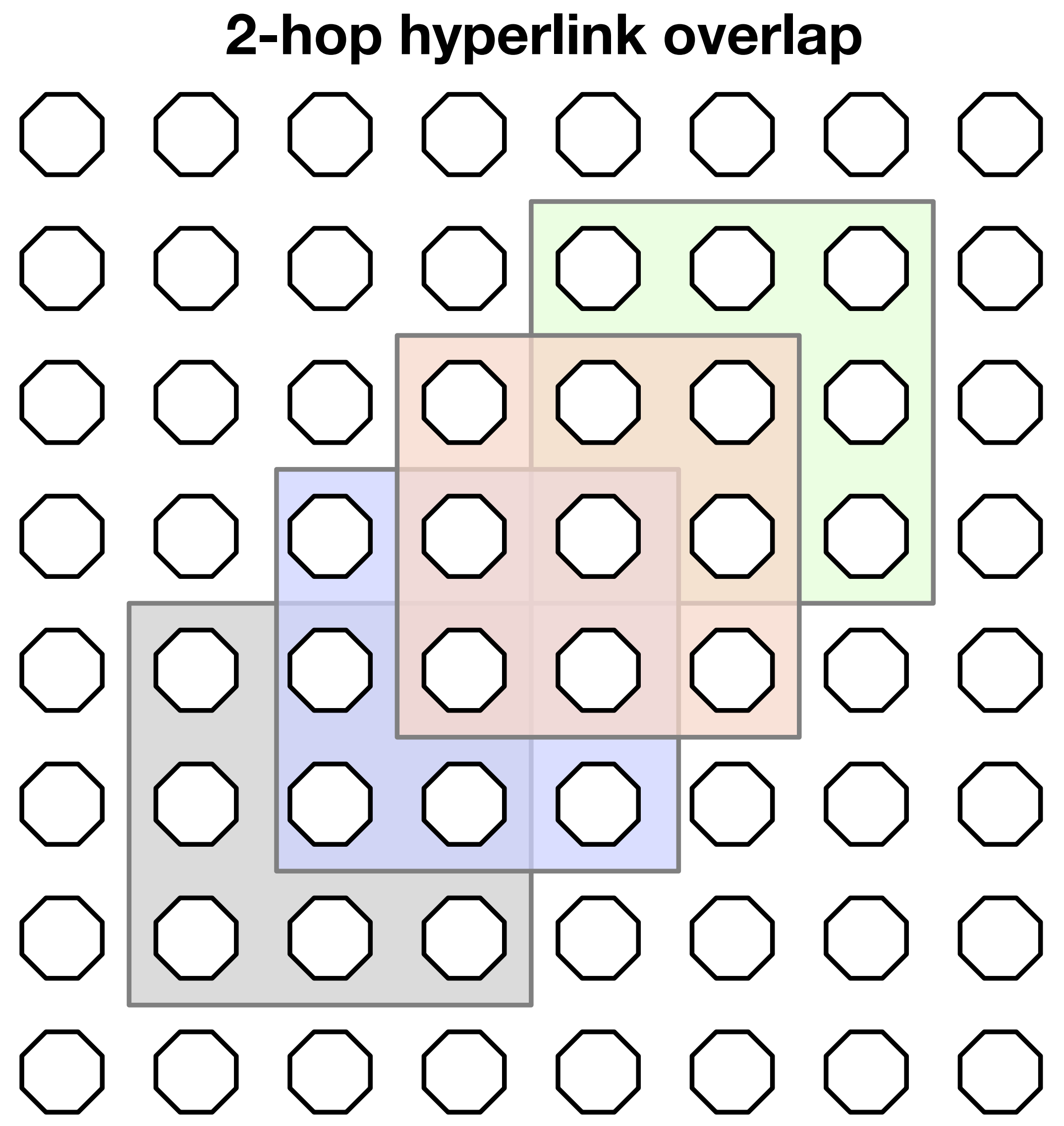

Liveness Tensor (for this Spekkens Qubit)
Charlie (Captain, Coordinator) sees and can coordinate the tokens of *all* the Alice's & Bob's in his cluster/tile
a1
a2
a4
b1
b2
b3
b4
C
Link 1
Link 2
Link 3
Link 4
a30
a31
a32
a33
…
Each hop in the chain is log(n) smaller data structure representing the knowledge balance for that link's liveness

Local topology awareness 1, 2 hops out, as basis for GVM, routing (and healing), and consensus engine. Demonstrate Raft and Paxos consensus protocols using single and multi- token Petri net. Demonstrate Reversible Constructor for 2PC, 3PC & 4PC





(a) 1st Hop Failure and Bypass (b) 2nd Hop Failure and Bypass (c) 3rd Hop Failure and Bypass (d) Hop Failure and Bypass

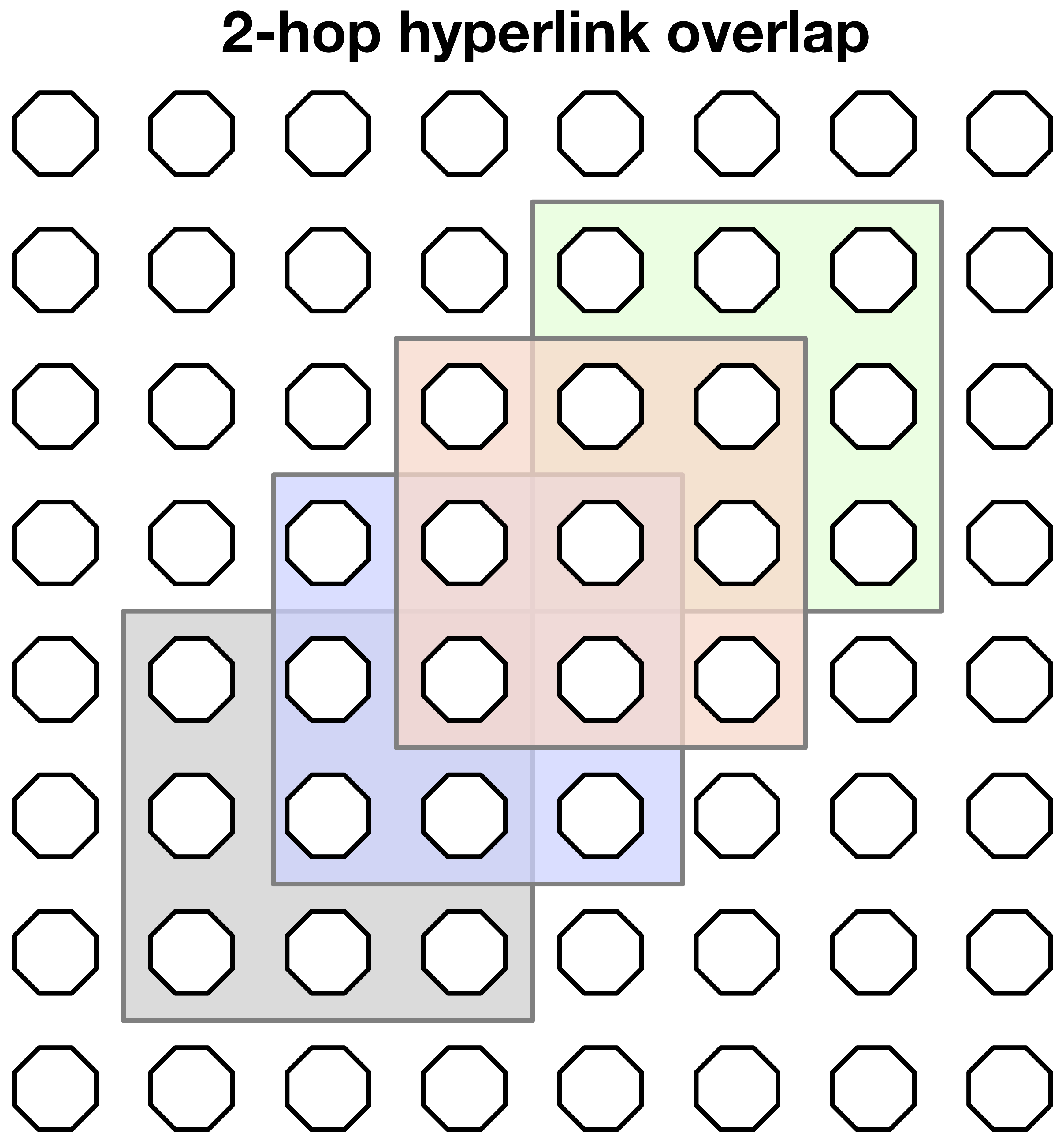


DÆ

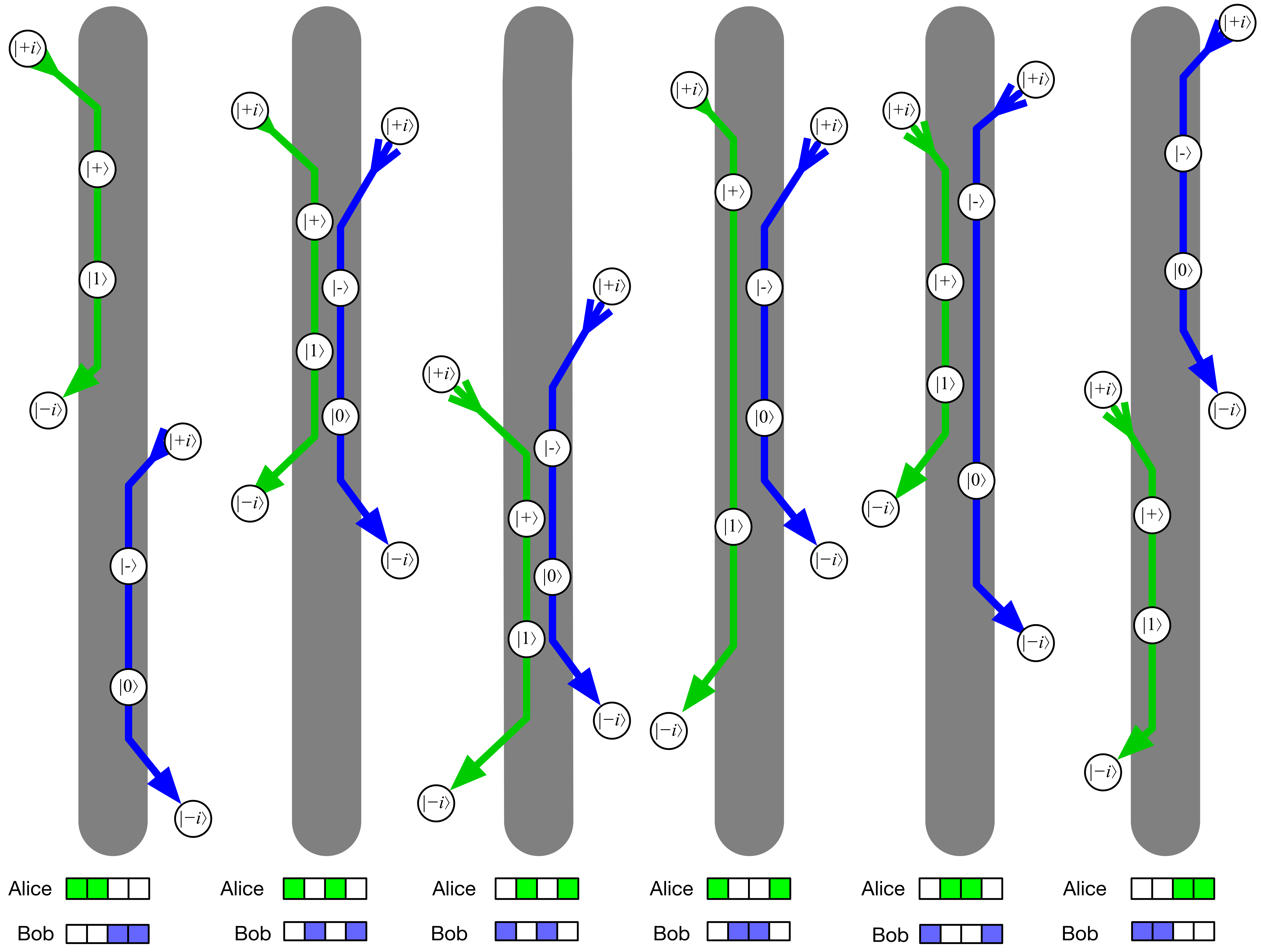
DÆDÆLUS

Liveness Tensor

- On the Link
- In the Cell
- In the Hypercell (8-ports)
- On the Trees
- On the Graph



Serialization in the FPGA SerDes

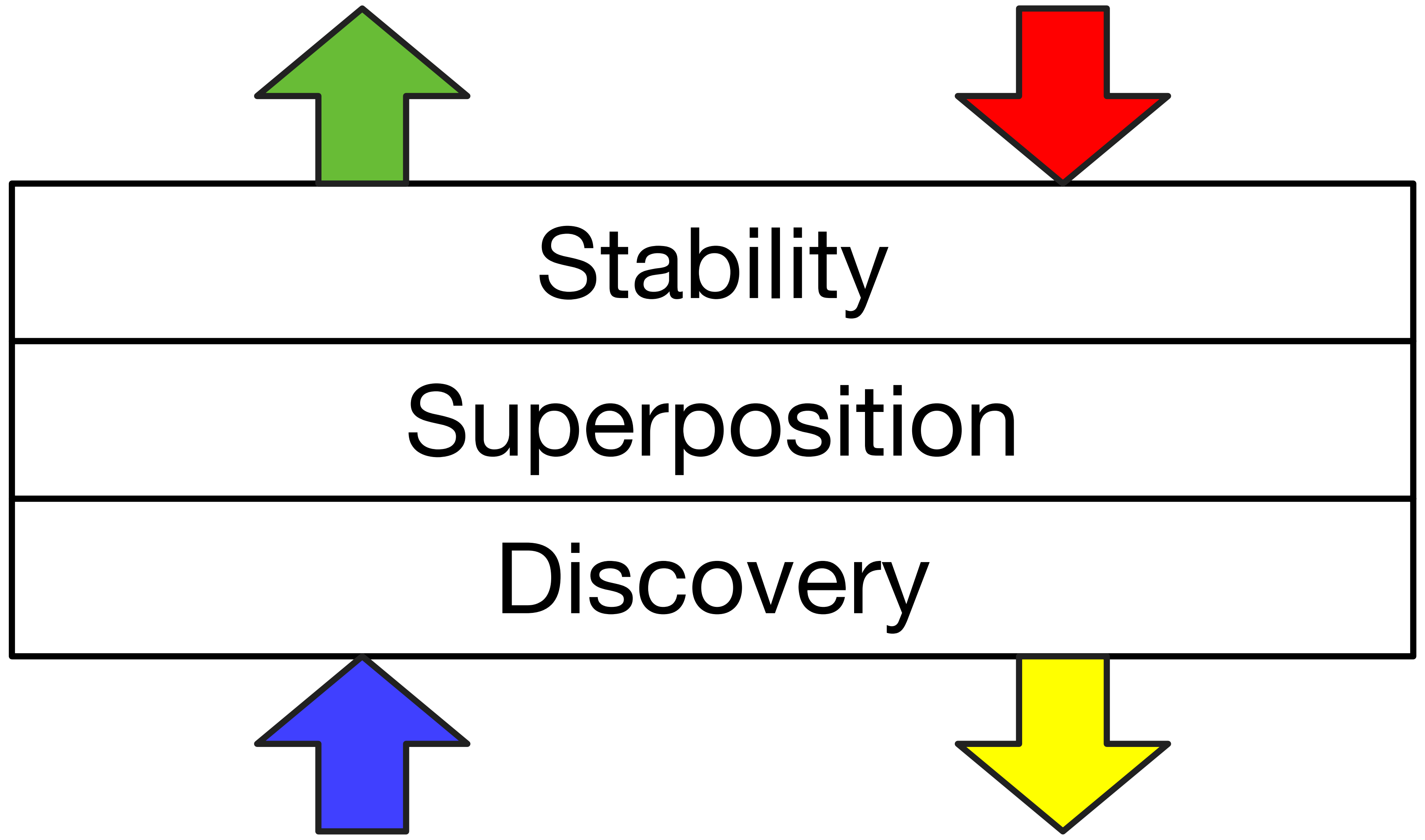


Petri-Spekkens Token Orderings in the FPGA SerDes and Disjoint Pairs

We proactively reorder events to match application constraints, eliminate race conditions, and prevent write skews before the database/ application sees them as an error

Causal Dynamics

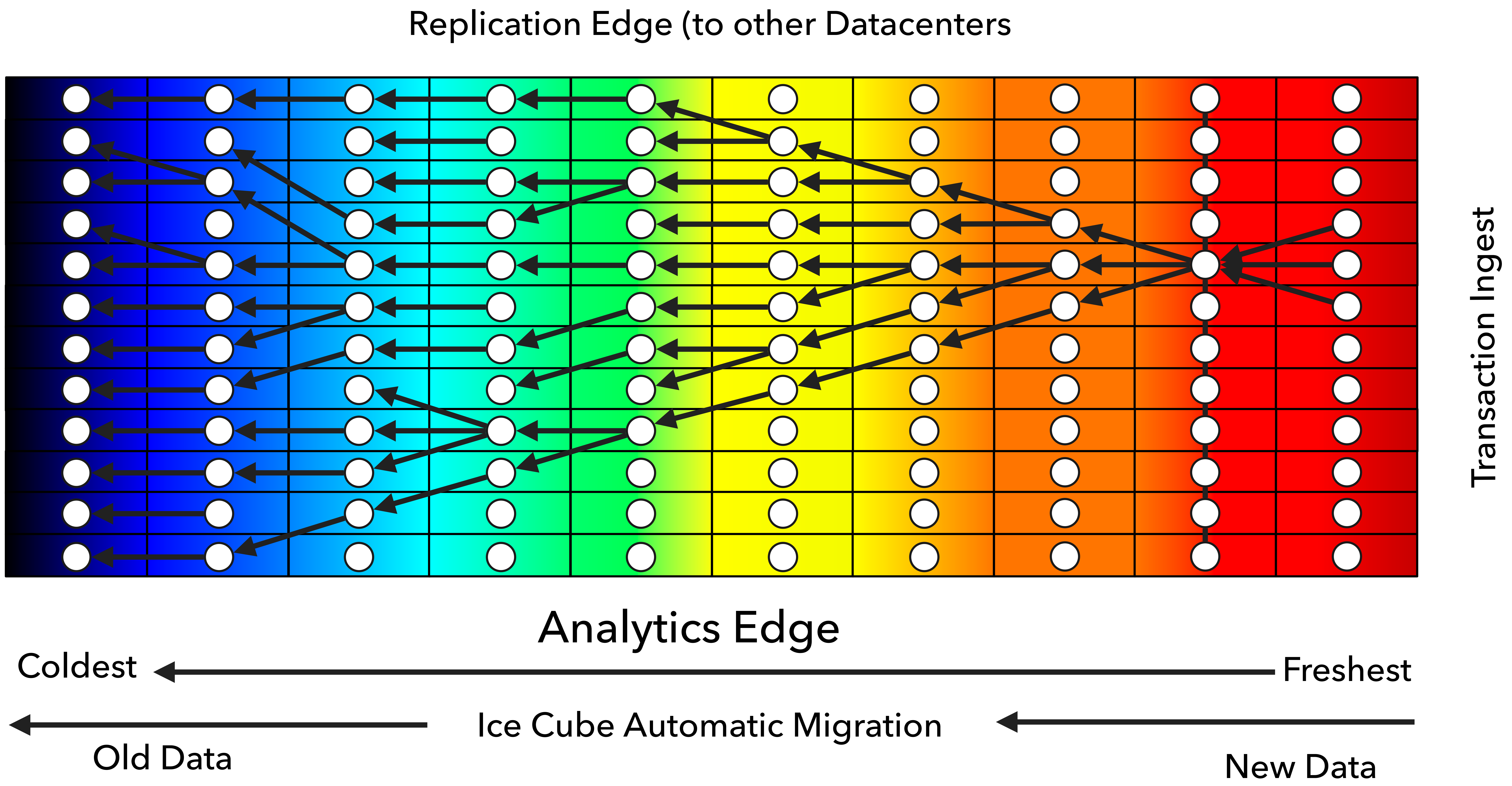
- An important goal of this design is to enable the setup, manipulation, and metastable strata of multi-level causal structures for Microservices. Jonathan Gorard, our resident expert, is cofounder of the Wolfram Physics Project
- We see an opportunity to revolutionize distributed systems in datacenters *everywhere*, by developing multiway algorithms to predictably 're-order causality' ***invisibly*** and ***indivisibly*** in the FPGA substructure (SmartNICs)



See Emily Adlam: Is There Causation in Fundamental Physics? New Insights from Process Matrices and Quantum Causal Modelling

OLTP Without Timestamps

Don't use Timestamps, use **Causal Trees**


Replication Edge (to other Datacenters
Transaction Ingest
Analytics Edge
Coldest
Freshest
Ice Cube Automatic Migration
Old Data
New Data

We Make Transactions Reliable

Reliable Substructure Clusters

Reversible transfers at line rates, with ultra-low Latency

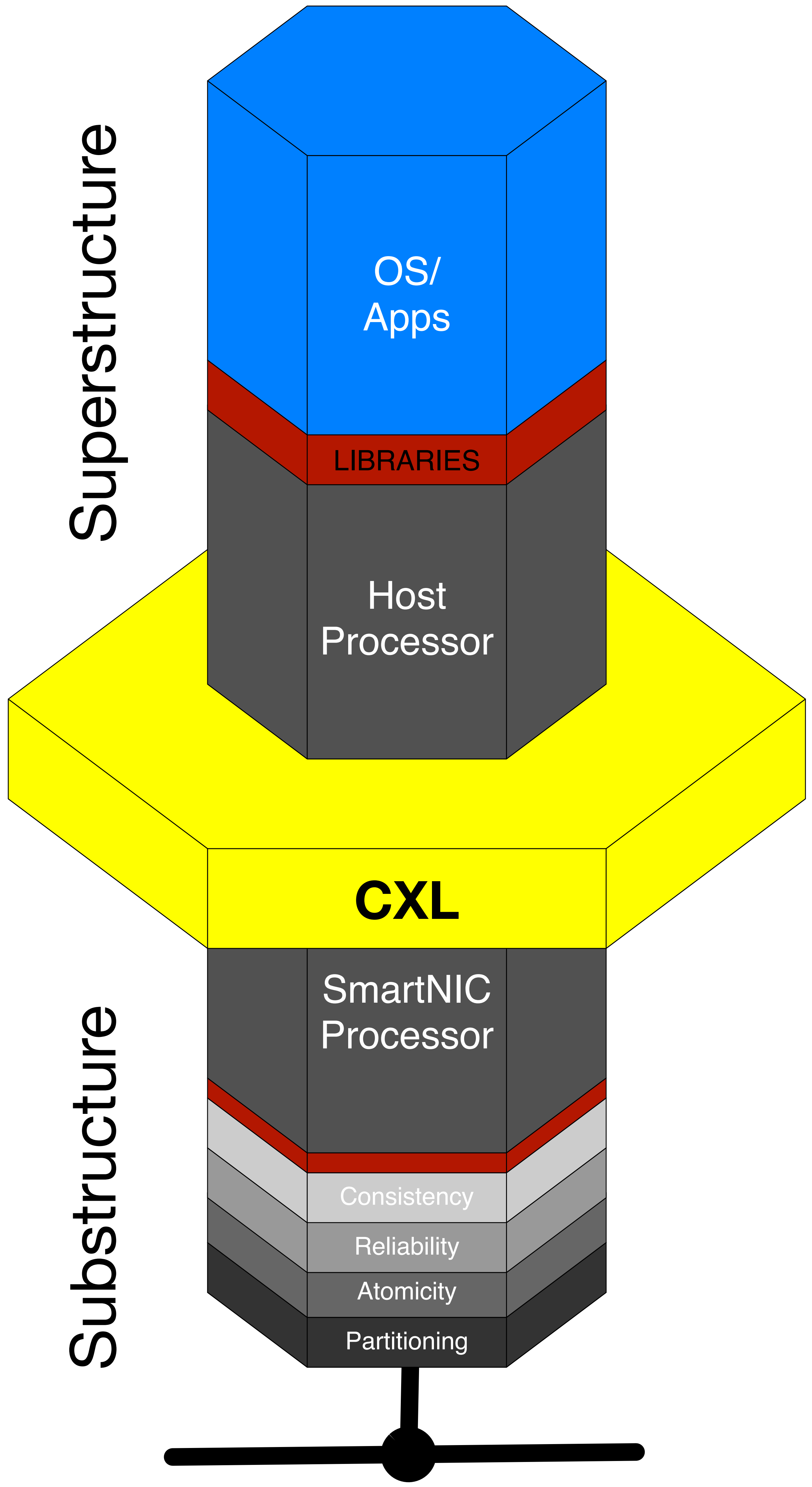
No tradeoff of Bandwidth and Latency

No Metastable Failures

FPGAs do not have a halt state!

They just run. Just circuits. Stuff goes in comes out, no halt states in between

Unlike ASICs, they don't take years to create



Engineering Notes and References

- There are three really important concepts here around "Shannon information", the transfer of Shannon information between nodes, and how error recovery is done when that transfer fails or is interrupted, so that certain properties are protected (conserved quantities, "I know that you know that I know", *Common Knowledge*).
- We understand how to design and build everything above. We also see opportunities to use Wolfram tools to explore the space of causal structures in ways that become standard industry toolchains for FPGA application fragments (specified by Petri nets):
- The only "research" aspect of this project is is the exploration of Causal Dynamics. For a 'current' entry into the causality literature see:

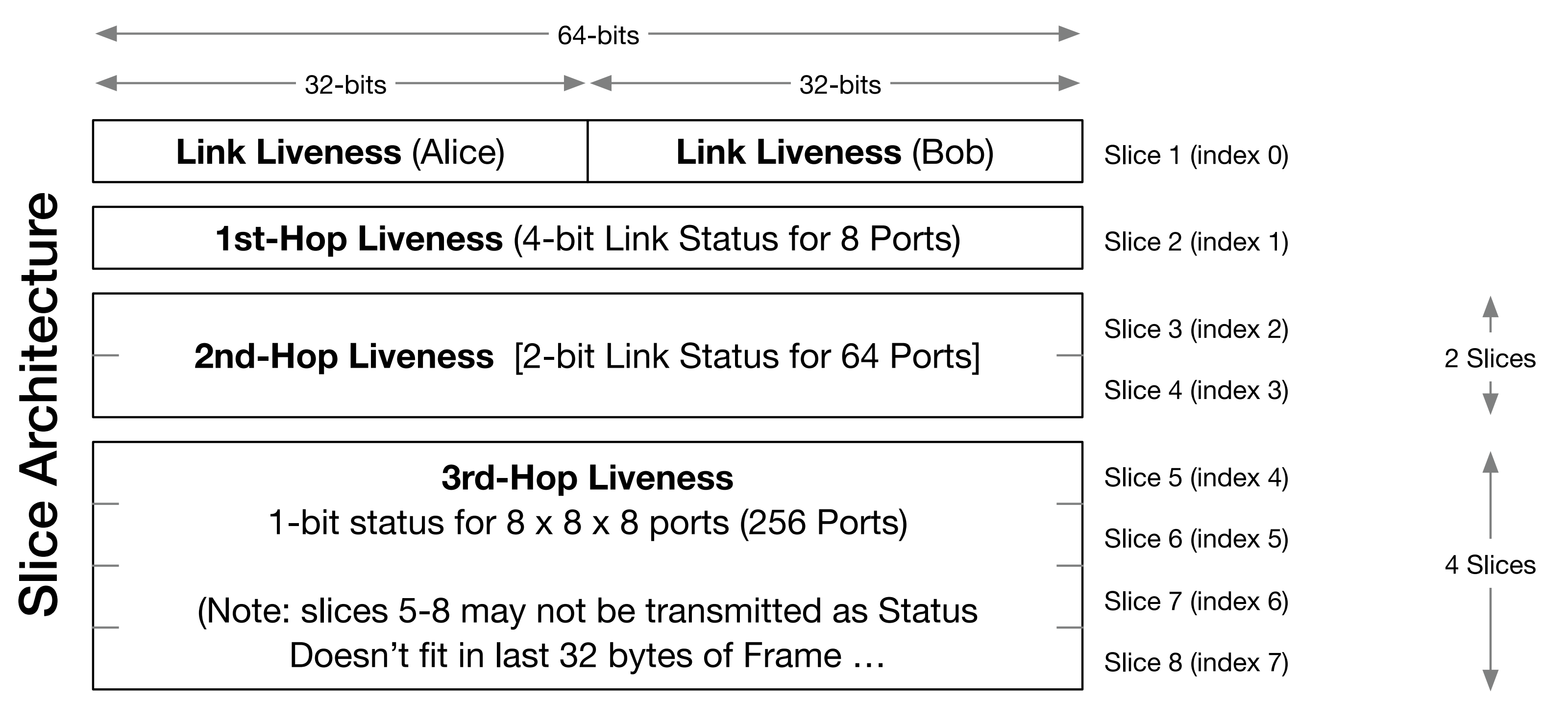
[1] Emily Adam: Is There Causation in Fundamental Physics? New Insights from Process Matrices and Quantum Causal Modeling. https://arxiv.org/pdf/2208.02721.pdf

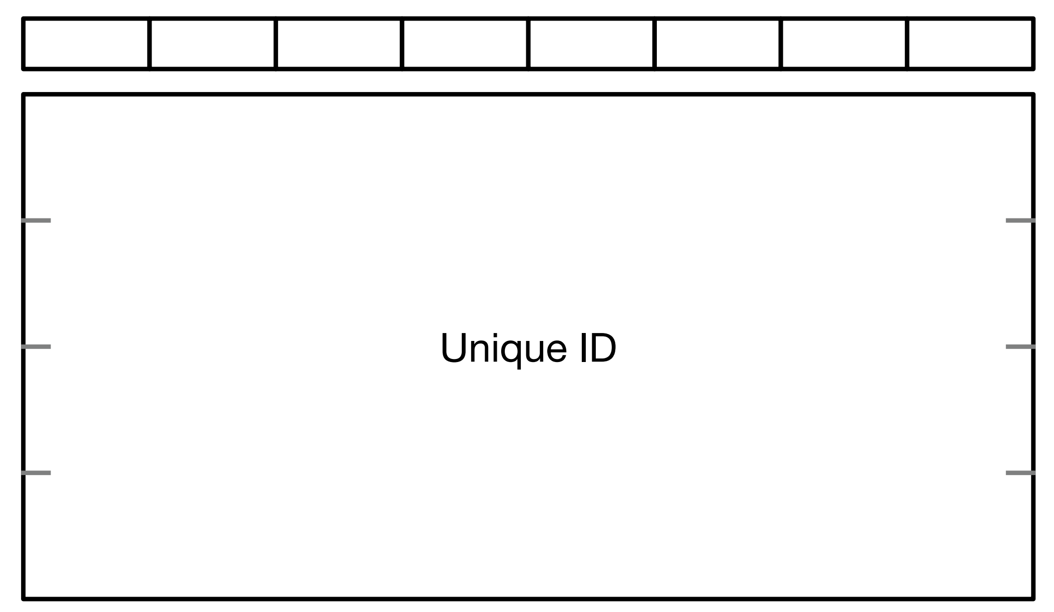
[2] Jordan Cotler et al, Information-theoretic Hardness of Out-of-Time-order Correlators. https://arxiv.org/abs/2208.02256

Protocol

- Timestamp-free event-only link protocol, with formal verification
- Address-free tree routing protocol, with local and remote asymmetry
- Reversible subtransactions within a reversible transaction
- Ethernet
    - Reliable Address-Free
    - Distributed Autonomous
    - Secure Address-Free

Slice Engine Architecture ECC





Slice 1 Link Liveness — Alice (Me) — Bob (Not Me)

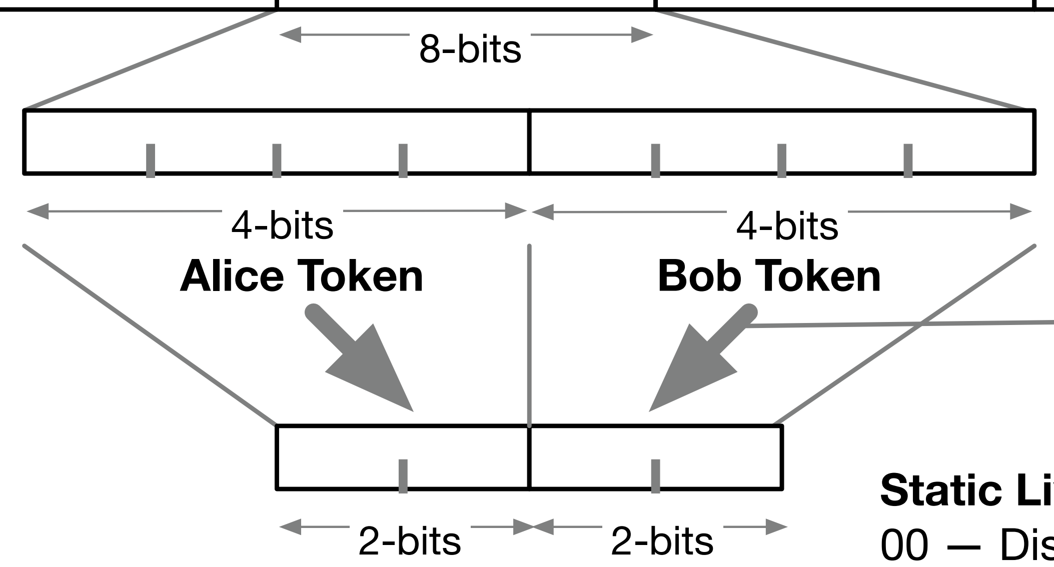
A1	A2	A3	A4	B1	B2	B3	B4
PROTOCOL SLICE	LINK LIVENESS	STATE MACHINE IDENTIFIER	STATE MACHINE TRANSITION	PROTOCOL SLICE	LINK LIVENESS	STATE MACHINE IDENTIFIER	STATE MACHINE TRANSITION

Slice 2 - 1-hop Liveness Tensor

We replace the "Link Liveness" for this link (which was already sent in the previous slice), with ECC, for when the frame is terminated early

Port 1	Port 2	Port 3	Port 4	Port 5	Port 6	Port 7	ECC

8 Ports - 1- Hop Liveness



Provides visibility into the current Link Status In the slice
(First hop Topology Awareness)

8 x 8 = 64 Ports - 2 - Hop Liveness

Alice Token — Bob Token

Static Liveness (Alice or Bob)	
00	Disconnected
01	Alice is Live
10	Bob is Live
11	Both Alice and Bob are Live

Dynamic Liveness (XOR)	
00	Disconnected
01	Alice is Live (transitory state)
10	Bob is Live (transitory state)
00	Both Alice and Bob are Live

Slice 3-6 - 2-hop Liveness Tensor

P11	P12	P13	P14	P15	P16	P17	P18	P21	P22	P23	P24	P25	P26	P27	P28	P31	P32	P33	P34	P35	P36	P37	P38	P41	P42	P43	P44	P45	P46	P47	P48
P51	P52	P53	P54	P55	P56	P57	P58	P61	P62	P63	P64	P65	P66	P67	P68	P71	P72	P73	P74	P75	P76	P77	P78	P81	P82	P83	P84	ECC			

1	2	4	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	28	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	ECC			

Next:

Continue Weekly 1/1's ?

1. Complete introduction Slides, Protocol, Implementation
2. Go through Applied Graph Theory (Chiplet Summit)
3. Grid presentation (Product, Features, Market)
4. Agree on Project: Tools to implement and verify protocols

Make it Work,

Make it Right,

Make it Perform,

Make it Presentable