AI for Good

Platform Limitations for 6G Real-Time Applications
Tail Latency, Determinism, and Architectural Alternatives



Presented by

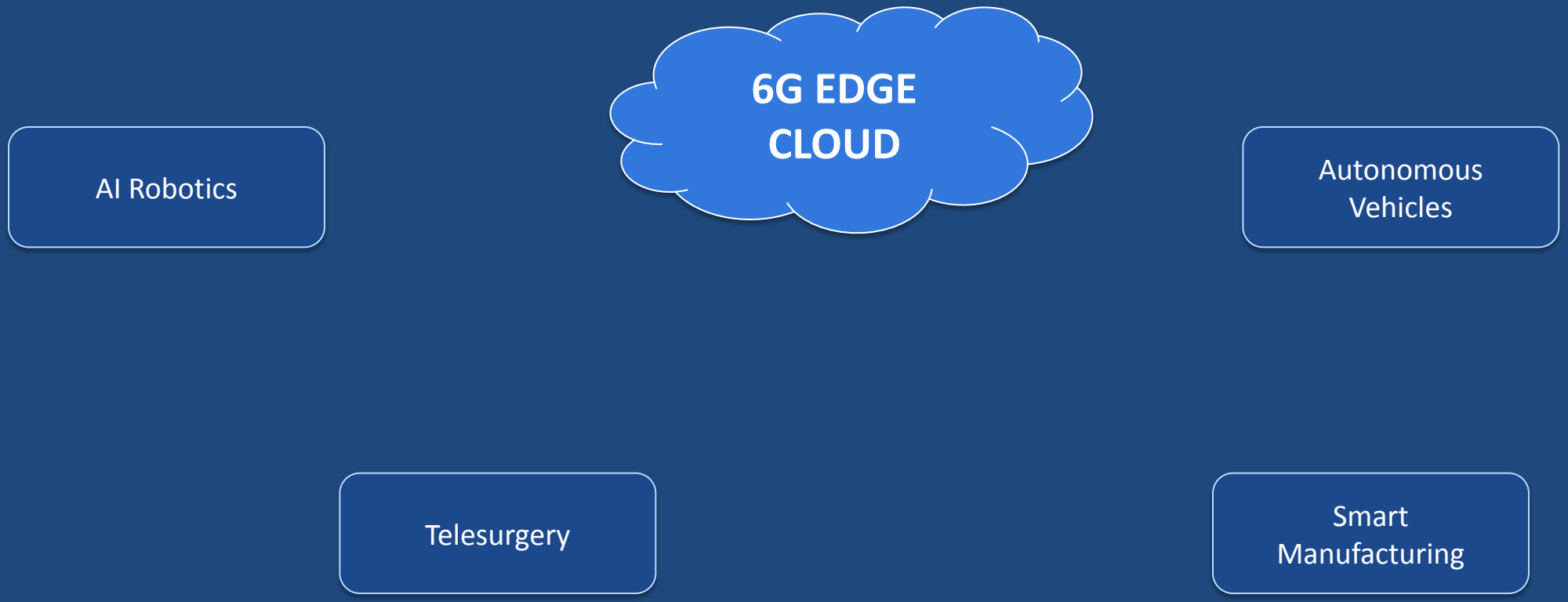
Michael Recchia

CEO/Chief Scientist

AdaptNet AI Laboratories



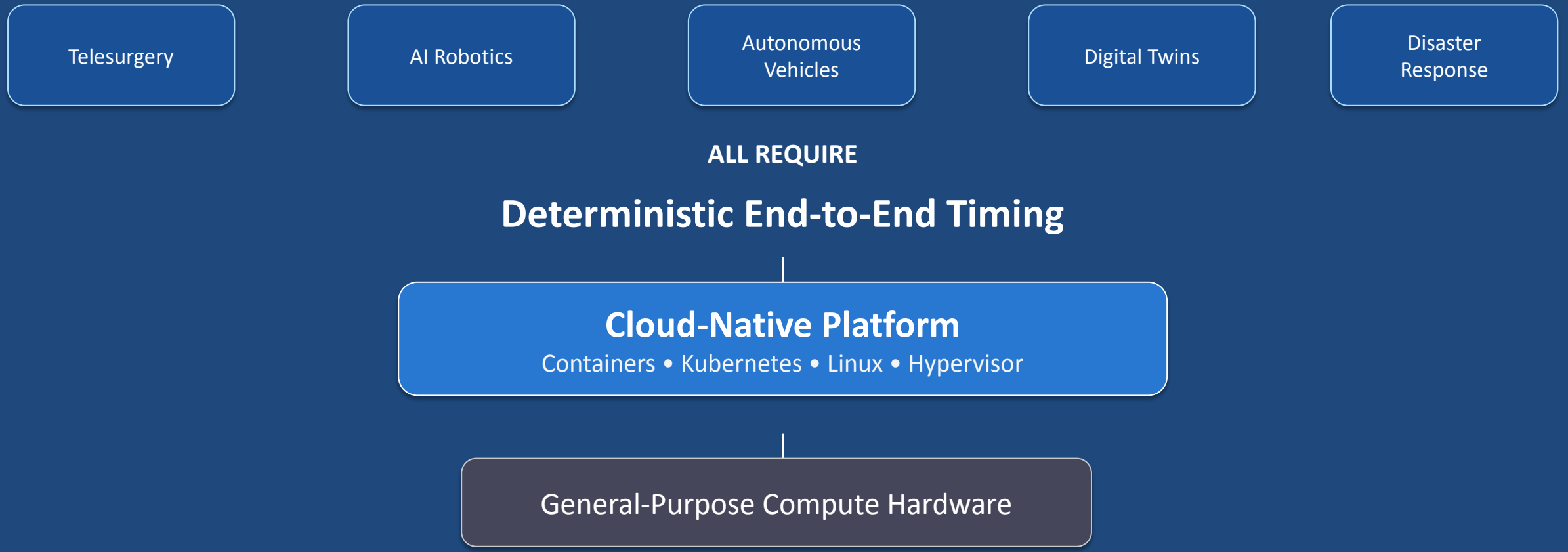
Platform Limitations of Today's Cloud-Native Computing Model for Deterministic 6G AI Applications



Future AI applications demand deterministic computing—not simply faster computing.

6G Changes the Computing Problem

The challenge is no longer throughput. The challenge is deterministic execution.



6G introduces applications requiring:

- Bounded worst-case latency
- Deterministic execution
- Microsecond-scale timing

The Cloud-Native Execution Path

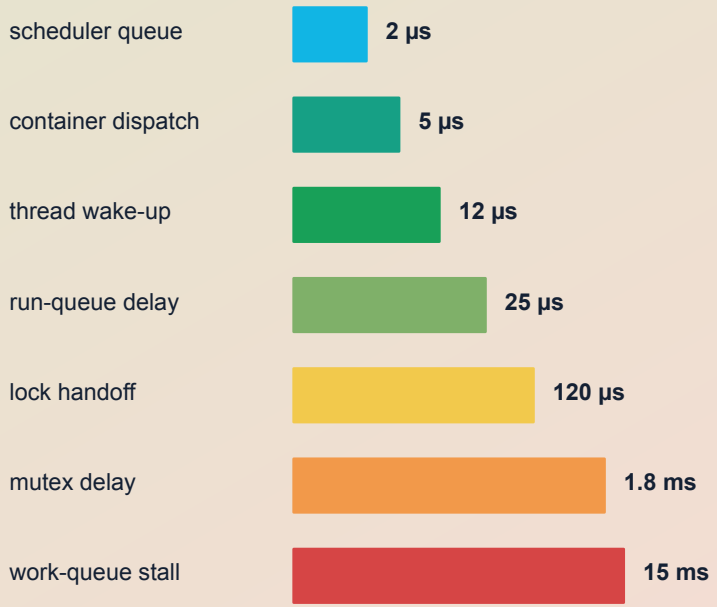
Every synchronization point introduces uncertainty into execution time.

COMPUTE STACK

- AI Application
- Container (CNF)
- Kubernetes Scheduler
- Linux Process Scheduler
- Worker Thread Pool
- Mutex
- Semaphore
- Shared Work Queue
- Critical Section

shared-state contention zone

ACCUMULATED WAIT STATES

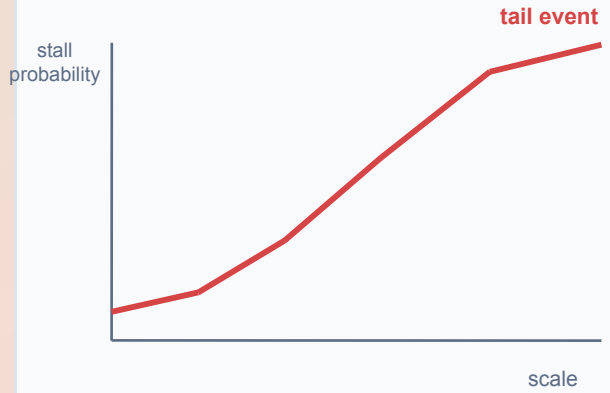


small waits × massive concurrency = millisecond-scale tail latency

TAIL-LATENCY THRESHOLD

One request may survive.

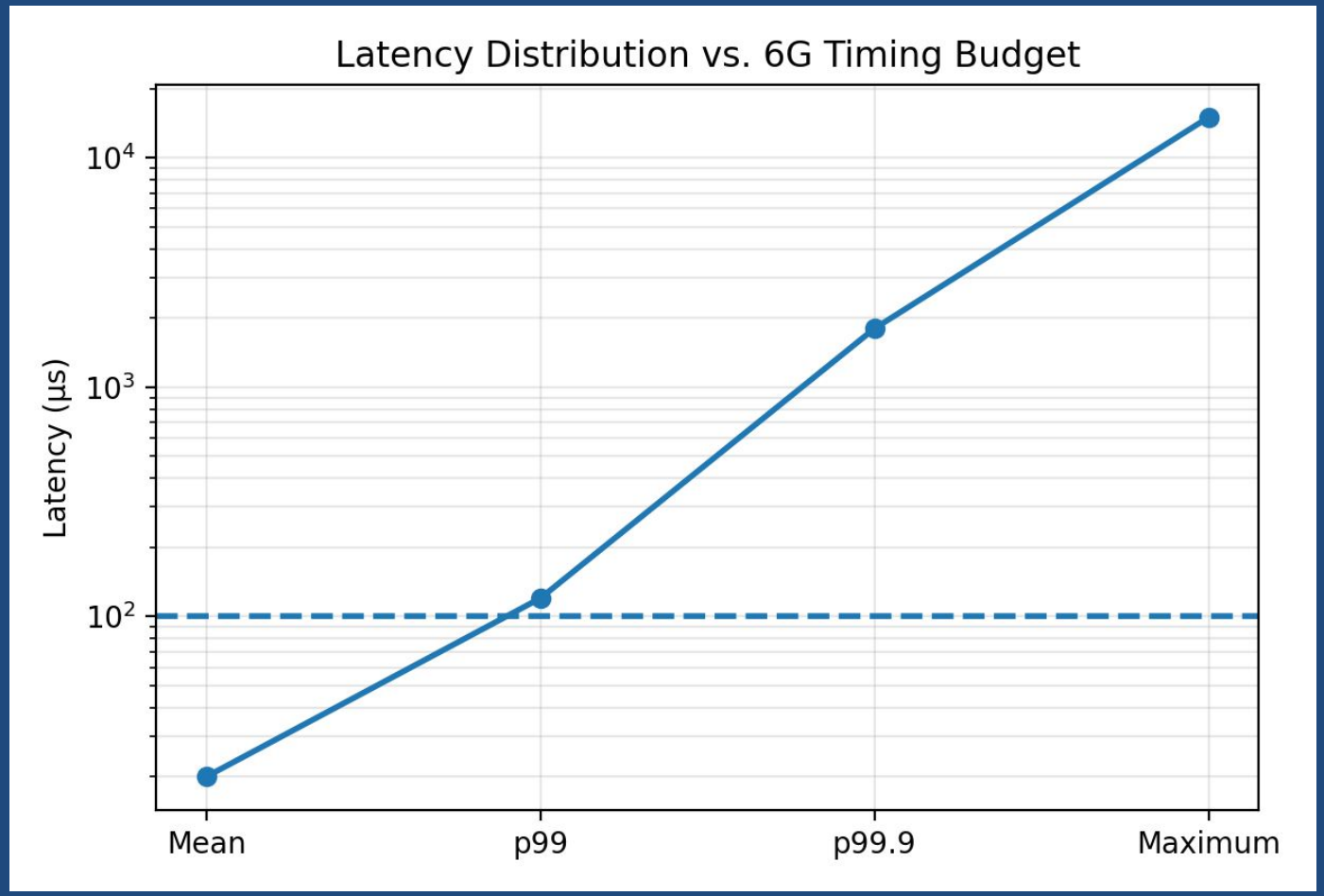
Millions of concurrent requests make stalls statistically unavoidable.



No individual synchronization event is catastrophic — the cumulative effect produces millisecond-scale tail latency.

Average Latency Tells the Wrong Story

6G fails because of worst-case latency—not average latency.



Timing Error Ratio

$$\frac{\text{Tail Latency}}{\text{Timing Budget}}$$

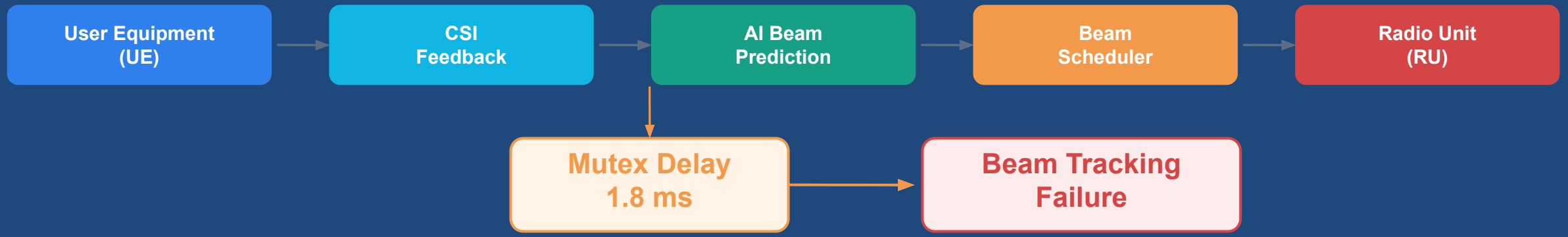
15 ms / 100 µs = 150×

150× OVER BUDGET

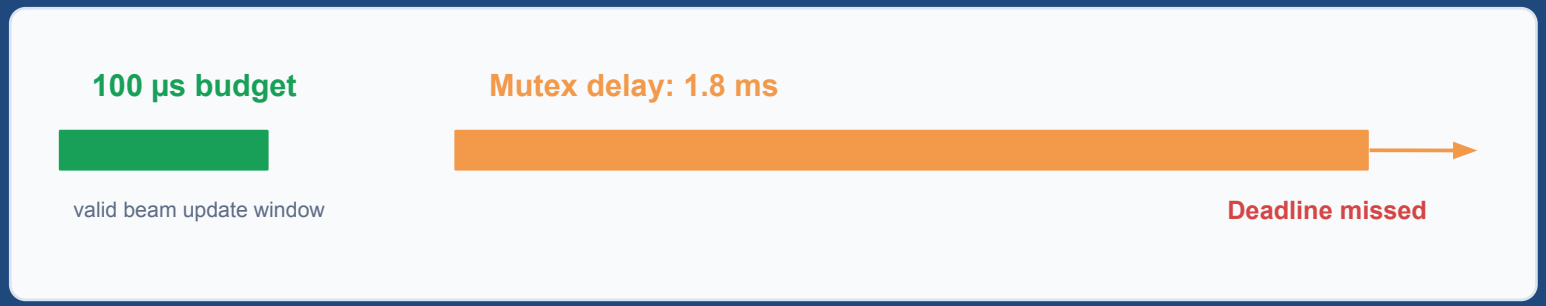
Systems fail because of rare events—not average events.

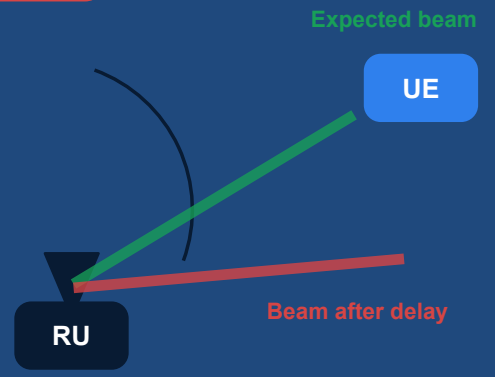
When Tail Latency Reaches the Radio

A delayed computation becomes a missed beamforming opportunity.



DETERMINISTIC RADIO DEADLINE





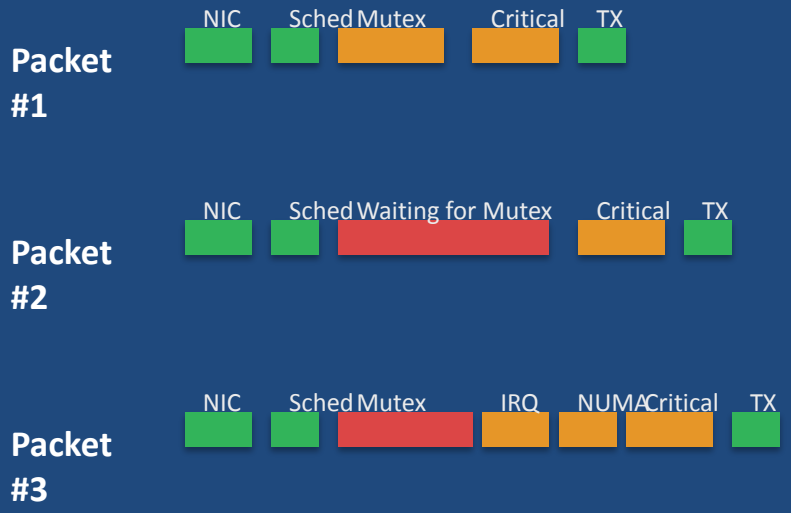
✖ Beam Missed

Deterministic deadlines expire before delayed beam updates can be applied — turning compute tail latency into radio failure.

Why Tail Latency Explodes

Latency is not additive—it compounds as synchronization increases.

Execution Timeline

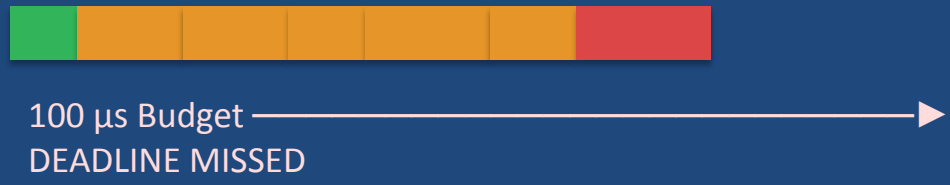


Tail Latency Model

L_{tail} = Scheduling + Mutex + Semaphore + IRQ + NUMA + Critical Section

$L_{tail} = \Sigma D_i$

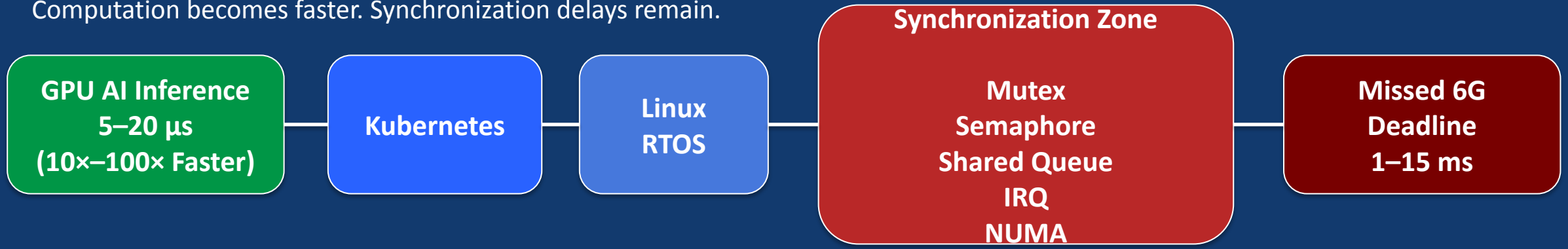
Latency Budget Consumption



Each delay is individually small. Their accumulation creates catastrophic worst-case latency.

Why Faster NVIDIA GPUs Cannot Solve Cloud-Native Tail Latency

Computation becomes faster. Synchronization delays remain.



Key Observation

- AI inference is no longer the dominant latency component.
- Every completed inference must still pass through Kubernetes scheduling, Linux scheduling, thread synchronization, mutexes, semaphores, interrupt handling, shared queues and NUMA contention.
- **Worst-case latency is therefore dominated by synchronization, not floating-point performance.**

$$L_{total} = L_{GPU} + L_{synchronization} \quad \text{where} \quad L_{GPU} \ll L_{synchronization}$$

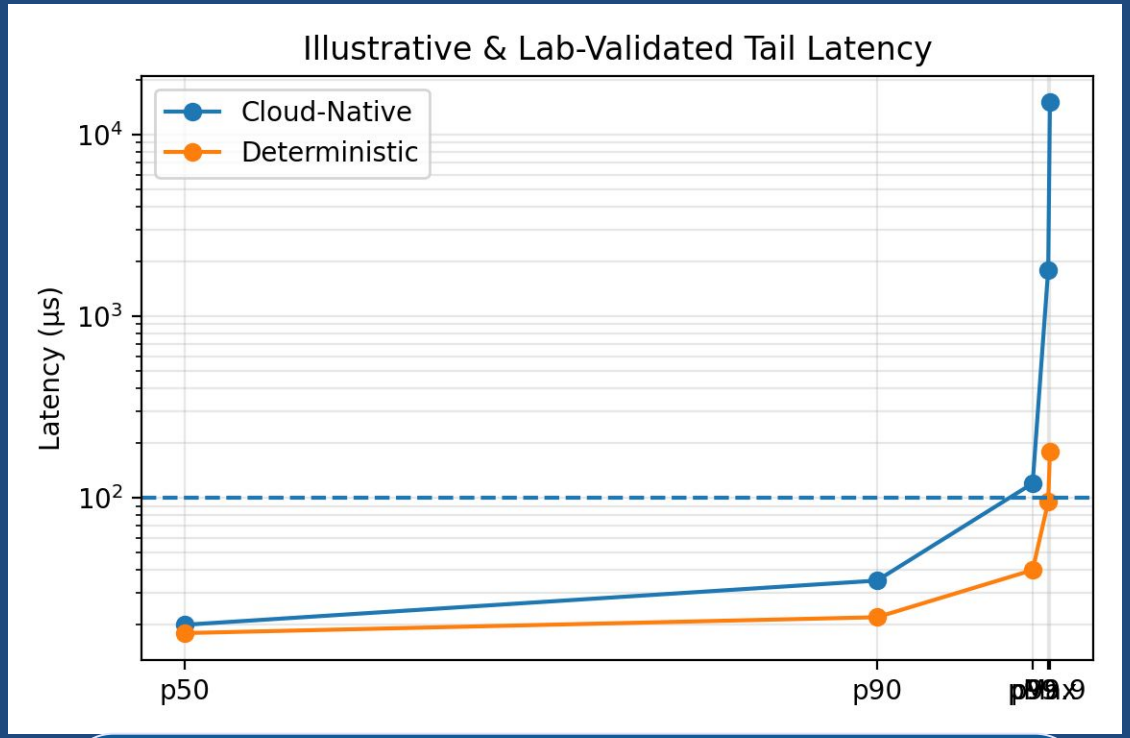
Faster processors cannot eliminate architectural synchronization bottlenecks, only a new computing architecture can GPUs accelerate computation. Deterministic architectures eliminate synchronization delay

What the Measurements Reveal

Average performance is nearly identical. Tail latency is fundamentally different.

Cloud-Native Execution

Packet
↓
Scheduler
↓
Mutex
↓
Semaphore
↓
Shared Queue
↓
Critical Section
↓
Transmit



Deterministic Pipeline

Packet
↓
Dedicated Core
↓
Lock-Free Ring
↓
Run-to-Completion
↓
Transmit

Illustrative & Lab-Validated Comparison

Mean: 20–25 µs vs. 18–22 µs (small difference)
Maximum: 15 ms vs. 180 µs (orders-of-magnitude difference)

The averages are similar. The worst-case behavior is not.

The Architectural Shift Required for Deterministic 6G

Cloud-native computing optimizes elasticity. 6G requires deterministic execution.

Elastic Computing (Today)

- Kubernetes
- ↓
- Shared Workers
- ↓
- Shared Queues / Mutexes
- ↓
- Linux Process Scheduler
- ↓
- General-Purpose CPU
- ↓
- Best-Effort Execution
- ↓
- Variable Tail Latency

Not an optimization problem
An architectural transition

Architectural Transition

Deterministic Computing (Future)

- Dedicated Processing Core
- ↓
- Lock-Free Ring Buffer
- ↓
- Run-to-Completion Pipeline
- ↓
- FPGA / SmartNIC / DPU
- ↓
- Hardware Time Synchronization
- ↓
- Deterministic Execution
- ↓
- Bounded Worst-Case Latency

Elasticity and Determinism Are Different Design Objectives

Toward Deterministic 6G Computing

One candidate architecture: The Appliance Computing Model (AdaptNet AI Labs)

AI / Network Orchestrator

Cloud Native: Move Containers
Appliance Model: Deploy Deterministic Appliances

Beamforming Appliance
- Dedicated Core
- Lock-Free Runtime
- FPGA Assist

AI Inference Appliance
- Deterministic Scheduler
- Specialized AI
- Bounded Timing

PHY Appliance
- Run-to-Completion
- Hardware Timing
- TSN Ready

Control Appliance
- Real-Time Control
- Predictable Execution
- Deterministic I/O

Elastic High-Speed Fabric • Data Center • Central Office • MTSO • Edge Cloud

General Computing → Virtual Machines → Containers → Deterministic Compute Appliances

Applications become deterministic appliances rather than elastic software instances.

Addendum: Sample Solution Alternatives

Today's Cloud-Native Computing

Future Deterministic Computing

6G Is Not Simply Faster Computing.

It Is Deterministic Computing.

The Platform Must Evolve.

The Future of AI Will Be Determined Not Only by Better Models...

...but by Deterministic Computing Architectures Capable of Guaranteeing Real-Time Execution

Research Vision: Composable Deterministic Computing Architectures for AI-Enabled 6G Networks

Slide 11 – Architectural Alternative #1: The Appliance Model

Dedicated Deterministic Functional Appliances for 6G Real-Time Systems

6G Service Request

Cloud vs Appliance
Shared → Dedicated
Dynamic → Static
Utilization → Determinism
Contention → Isolation

High-Speed Deterministic Fabric

AI/ML Appliance	RAN CU Appliance	User Plane Appliance	Security Appliance	Analytics Appliance
Dedicated CPU/GPU/DPU Static Resources Lock-Free Execution	Dedicated CPU/GPU/DPU Static Resources Lock-Free Execution	Dedicated CPU/GPU/DPU Static Resources Lock-Free Execution	Dedicated CPU/GPU/DPU Static Resources Lock-Free Execution	Dedicated CPU/GPU/DPU Static Resources Lock-Free Execution

Key Design Principles

- Dedicated compute per function
- Static resource allocation
- Minimal shared state
- Lock-free execution
- Independent scaling
- Deterministic latency

Potential Benefits

✓ Reduced mutex contention
✓ Predictable worst-case latency
✓ Simpler timing analysis
✓ Easier certification
✓ Fault isolation
✓ Reduced scheduler interference

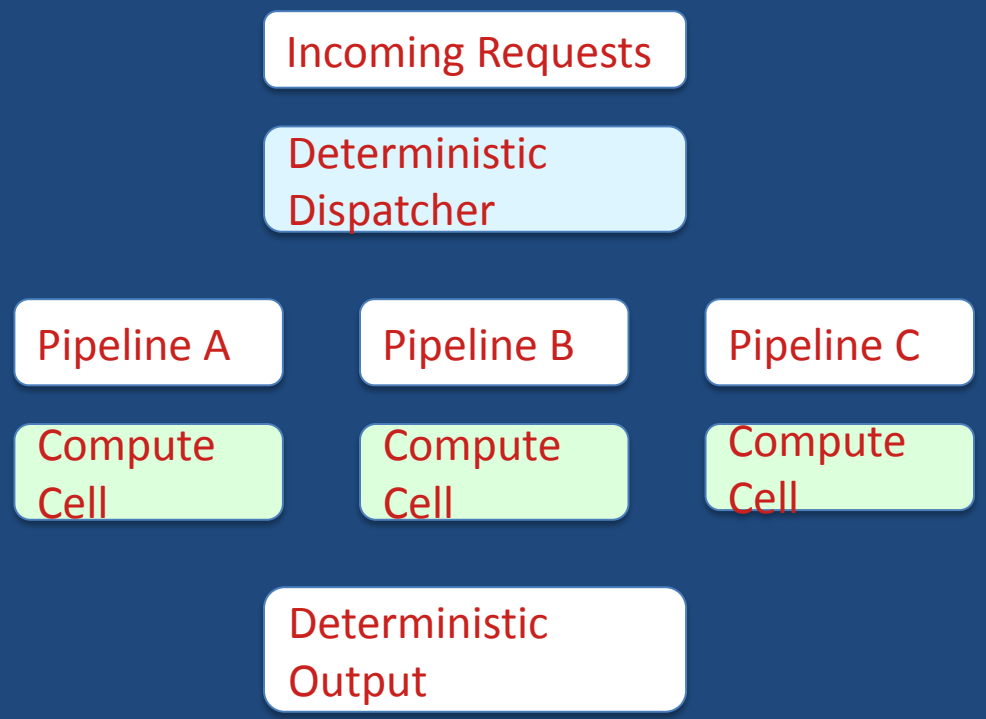
Slide 12 – Architectural Alternative #2: Deterministic Execution Fabric

Replace Dynamic Resource Scheduling with Deterministic Execution Pipelines

Traditional Cloud-Native

- Incoming Requests
- Kubernetes Scheduler
- Linux Scheduler
- Shared CPUs
- Mutexes
- Shared Memory
- Variable Latency

Deterministic Execution



Cloud vs Fabric

Schedule→Route
Shared→Dedicated
Dynamic→Fixed
Mutexes→Ownership
Variable→Predictable

Design Principles

- Static execution ownership
- CPU affinity by design
- Memory locality
- Lock-free communication
- Deterministic dispatch
- No centralized scheduler
- No shared execution queues

Expected Benefits

- ✓ Lower p99 & p99.9 latency
- ✓ Reduced jitter
- ✓ Better cache locality
- ✓ Elimination of scheduler contention
- ✓ Improved scalability
- ✓ Predictable timing

Slide 13 – Architectural Alternative #3: Specialized Learning Spheres

A Self-Organizing Deterministic Computing Architecture

Incoming Requests

Intelligent Domain Classifier (Neural Tree)

Learning Sphere A

Specialized Neural Network

Dedicated Compute Cells

Learning Sphere B

Specialized Neural Network

Dedicated Compute Cells

Learning Sphere C

Specialized Neural Network

Dedicated Compute Cells

Coordinated Global Response

Conventional vs Learning Spheres

General→Specialized
Shared→Dedicated
Central→Independent
Dynamic→Deterministic
Generic→Domain

Core Concepts

- Neural Trees classify work
- Independent specialized learning
- Dedicated execution resources
- Minimal shared state
- Domain-specific optimization
- Distributed learning

Potential Benefits

✓ Reduced synchronization
✓ Lower contention
✓ Continuous specialization
✓ Improved determinism
✓ Horizontal scalability
✓ Adaptive optimization