LECTURE SERIES

ADVANCED COMPUTER SCIENCE

MODULE 04

Scalable Networks

SYNAPSE INSTITUTE OF TECHNOLOGY

Data-Driven Architecture & Distributed Systems

Patterns for Scalability, Consistency, and Resilient Storage

LECTURER

Dr. Erik Lindqvist

Chair of Distributed Systems

ACADEMIC TERM

Autumn Semester 2026

Course Code: CS 402

Lecture Agenda

Key Topics & Session Structure

Part I: Architectural Foundations

- Monolith vs. Microservices
- Fallacies of Network Computing
- CAP Theorem & PACELC Trade-offs

Part II: Storage & Caching

- Multi-Tier Data Caching (Redis)
- Write-Through vs. Write-Back
- Database Sharding Strategies

Part III: Production Case Study

- Meridian Platform Traffic Spikes
- 1.2M Concurrent Connections
- Empirical Latency Metrics

Part IV: Resiliency Engineering

- Circuit Breakers & Backoff
- Distributed Consensus (Raft Engine)
- Summary & System Checklist

MODULE 01

Architectural Foundations

Deconstructing Distributed Complexity & Consistency Guarantees

System Paradigms

Comparing Monolithic and Distributed Microservices

Monolithic Architecture

Single unified codebase and deployment artifact.

- **Memory Access:** In-process calls ($< 1\,\mu s$).
- **Transactions:** ACID via single RDBMS.
- **Scaling:** Vertical scale-up (CPU/RAM).
- **Deployment:** Tightly coupled releases.

Distributed Microservices

Decoupled services over network RPC.

- **Network RPC:** Remote API calls (5–50 ms).
- **Transactions:** BASE / Eventual consistency.
- **Scaling:** Horizontal scale-out (elastic).
- **Deployment:** Independent CI/CD pipelines.

Key Insight: Transitioning to microservices trades **code complexity** for **operational complexity**. Network topology becomes a primary design constraint.

The CAP Theorem & PACELC Framework

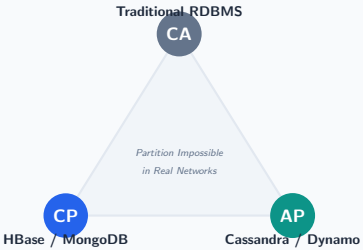
Fundamental Trade-offs in Distributed Data

- **Consistency (C):** Every read receives the most recent write or an error.
- **Availability (A):** Every non-failing node returns a non-error response.
- **Partition Tolerance (P):** System functions despite network dropouts.

The PACELC Extension:

If Partition (P): Choose Availability (A) vs. Consistency (C).

Else (E): Choose Latency (L) vs. Consistency (C).



Fallacies of Distributed Computing

Common Architectural Pitfalls in Production

Network Assumptions

1. **Network is reliable:** Packets drop, routers fail.
2. **Latency is zero:** Round-trips dominate execution.
3. **Bandwidth is infinite:** Saturation degrades nodes.

Topology Assumptions

1. **Network is secure:** Internal traffic needs auth.
2. **Topology is static:** Containers scale dynamically.
3. **Transport cost is zero:** Serialization takes CPU.

Architectural Rule: Always design for asynchronous failure. Assume network requests will fail, timeout, or arrive out of order.

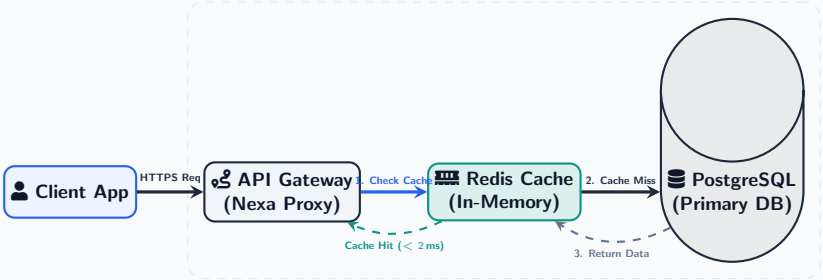
MODULE 02

Storage Systems & Caching Strategies

Optimizing Memory Hierarchies and Database Invalidation

Multi-Tier Storage Architecture

Data Flow Through In-Memory Caching and Persistent Storage



L1 In-Memory Cache Tier

Delivers sub-millisecond read performance for hot data keys (95% hit target rate).

Persistent Storage Tier

Ensures ACID durability for critical state with write-ahead logging (WAL).

Cache Consistency Patterns

Comparative Analysis of Invalidation Strategies

Pattern	Write Flow	Latency Profile	Primary Use Case
Cache-Aside	App writes to DB; invalidates key.	Read penalty on miss; low write latency.	General web apps, dynamic feeds.
Write-Through	App writes to Cache; Cache writes DB synchronously.	Higher write latency; guaranteed consistency.	Financial ledgers, transactions.
Write-Back	App writes to Cache; async write to DB.	Lowest write latency; risk of data loss.	Analytics & telemetry logs.
Refresh-Ahead	Cache auto-refreshes hot keys prior to TTL.	Optimal read latency; complex setup.	High-traffic static catalogs.

Phil Karlton's Adage: *"There are only two hard things in Computer Science: cache invalidation and naming things."*

MODULE 03

Production Case Study: Meridian Cloud

Real-World High-Concurrency Load Handling & Scaling Metrics

Case Study: Meridian Cloud Platform

Handling Flash Traffic Spikes during Global Events

The Challenge

- Peak load burst: **1.2M req/sec** during event.
- Primary DB CPU saturated at **99.4%**.
- Cascading failures across microservices.

- Implemented **Rate Limiting** via Redis.
- Deployed **Consistent Hashing** on storage.
- Enforced queueing via **Vertex Message Bus**.

Outcome: System stabilized with zero database crashes and average response time reduced from **4,200 ms** to **38 ms** at 99th percentile.

Empirical System Metrics

Post-Optimization Performance Benchmarks at Meridian

-98.2%

Latency Reduction

P99 latency dropped from 4.2s to 38ms post-caching.

99.999%

High Availability

Five-nines uptime maintained across 90-day window.

4.5x

Throughput Gain

Sustained requests/sec increased without added hardware.

Key Observation: Caching and queueing decoupled compute spikes from persistent storage, converting catastrophic failures into controlled queue latency.

MODULE 04

Resiliency Engineering & Summary

Fault Isolation, Graceful Degradation, and Architectural Guidelines

Circuit Breaker Pattern

Preventing Cascading System Failures

State Machine Lifecycle

- **Closed State:** Normal operations. Traffic flows freely to downstream service.
- **Open State:** Error threshold exceeded ($> 15\%$). Calls fail fast immediately.
- **Half-Open State:** Trial period. Probes service with test traffic.

```
if (errors > threshold) {
openBreaker();
return fallbackData();
} else {
executeServiceCall();
}
```

Combined Pattern:

Pair Circuit Breakers with **Exponential Backoff** and randomized **Jitter**.

Architectural Checklist

Design Principles for Distributed Systems

System Design Musts

- **Idempotency:** Ensure API writes can be retried with UUID.
- **Stateless Services:** Offload session state to Redis.
- **Graceful Degradation:** Serve cached data if primary fails.

Operational Anti-Patterns

- **Distributed Transactions:** Avoid 2PC across microservices.
- **Hardcoded Endpoints:** Utilize dynamic Discovery (K8s).
- **Shared Databases:** Never allow services to share raw DB.

Golden Rule: Prioritize simplicity. Distributed systems are inherently complex; avoid adding unnecessary distribution unless required by scale.

Thank You!

Questions & Discussion

Office Hours: Tuesdays 14:00–16:00 — Campus Building 04, Room 302