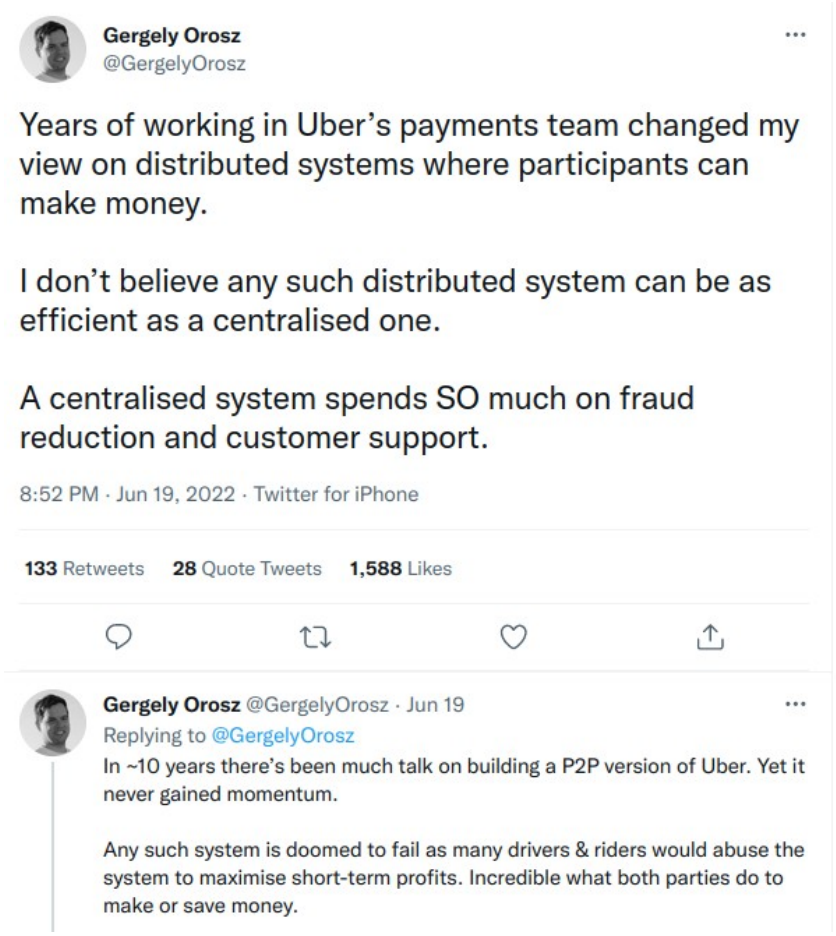
Introduction to Distributed Systems

Prof. Walter Kriha, fall term 2022 (last DS lecture)

Hochschule der Medien

A Bit of Motivation...



Gergely Orosz
@GergelyOrosz

Years of working in Uber's payments team changed my view on distributed systems where participants can make money.

I don't believe any such distributed system can be as efficient as a centralised one.

A centralised system spends SO much on fraud reduction and customer support.

8:52 PM · Jun 19, 2022 · Twitter for iPhone

133 Retweets **28** Quote Tweets **1,588** Likes

Gergely Orosz @GergelyOrosz · Jun 19
Replying to @GergelyOrosz
In ~10 years there's been much talk on building a P2P version of Uber. Yet it never gained momentum.

Any such system is doomed to fail as many drivers & riders would abuse the system to maximise short-term profits. Incredible what both parties do to make or save money.

Gergely Orosz @GergelyOrosz · Jun 19
What changed my belief is seeing all the things drivers do to make money (using every single arbitrage opportunity - eg forcing riders to cancel & collect the cancel fee, as this is more profitable at eg airports) and also riders (from money laundering to car theft etc).

Wild...

Looks like there is more to it than just tech...

Overall Goals

- Learn the basic concepts of Distributed Systems like concurrency and remoteness, consensus and failure models.
- Understand different programming models for Distributed Systems
- Acquire a theoretical foundation of computability in distributed systems
- Learn to design distributed systems with a focus on performance, availability, scalability and security
- Understand the constraints imposed by hardware and failures
- Learn what it takes to BUILD middleware for DS

Goal for today

- Give an overview of distributed systems. Later lectures will dig into the gory details like security, transactions, remote calling mechanisms etc.

Introduction

- What is a Distributed System (DS)
- What makes it difficult for Developers?
- Why distribute?
- Examples of DS
- Characteristics of DS
- Middleware for DS
- Concepts and Architectures (Scale, Parallelism, Latency etc.)
- Resources

Definition of a Distributed System:

Independent agents repeatedly interacting in a way that a coherent behavior („system“) emerges.
Events happen concurrently and parallel.

Agents are dumb or intelligent, with incomplete local information, differ in capabilities, suffer from random events (local or network). See: A.B.Downey, Think Complexity

Emergence

Strong Emergence: We cannot predict what will emerge (Chalmers, game of life)

Weak Emergence: Things are combined by simple principles but the result surprises (flock of birds)

Evolutionary Emergence: from egg to human being: Complex but robust

Constructed Emergence: e.g. Distributed Systems: Complex but often NOT robust

Emergent Failure Modes: Cascading Failures in Constructed Emergence

More: K.Mitchell, http://www.wiringthebrain.com/2022/05/the-riddle-of-emergence-where-do-novel.html

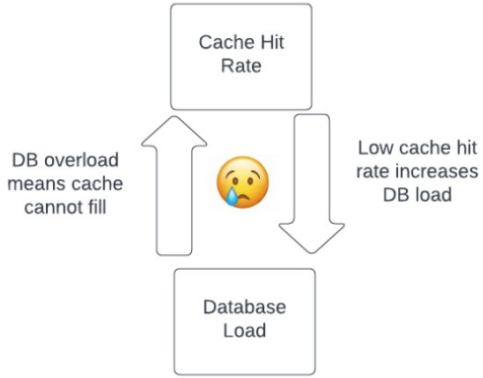
Emergent Failure Mode Example

Slack's Incident on 2-22-22

Double Trouble with Datastores

Laura Nolan Senior Staff Engineer

What caused us to go from a stable serving state to a state of overload? The answer turned out to lie in complex interactions between our application, the Vitess datastores, caching system, and our service discovery system.



More: https://slack.engineering/slacks-incident-on-2-22-22/
You should check out Laura Nolan's talks on youtube.
Theory: Metastable Failures in Distributed Systems
N. Bronson, https://sigops.org/s/conferences/hotos/2021/papers/hotos21-s11-bronson.pdf

Why is a DS difficult for Devs?

Emergence

Single Machine View

Errors are the Fabric

There is NO free Lunch

It is total end-to-end System Engineering

Finally, all distributed systems algorithms are based on the failures we expect and how we treat them. It took me quite a while to really understand this fundamental point….

Single Machine View

"Developers typically view the world through their IDE, from the perspective of a single machine. They default to reasoning about a system through the lens of the code that is immediately in front of them. Although they are consciously aware of the network as a source of failure, concurrency, asynchrony and latency, their porthole view does not necessarily cause them to take a step back and appreciate what they consider situational – a problem in this case – is actually foundational; that it arises from the nature of the system."

Kevin Henney, https://www.infoq.com/news/2022/09/distributed-system-knowable/?itm_source=infoq&itm_medium=popular_widget&itm_campaign=popular_content_list&itm_content=

Errors are the “fabric” of a DS

“Although exception handlers might acknowledge failures, they don’t reflect their normality. I’ve seen more than one coding guideline state that "exceptions should be exceptional". This advice is little more than wordplay – it’s neither helpful nor realistic. There is nothing exceptional about timeouts, disconnects and other errors in a networked environment – these are "business as usual".

These "little issues" are also not little: they shape – in fact, they are – the fabric from which a distributed system is made.”

Kevin Henney, https://www.infoq.com/news/2022/09/distributed-system-knowable/?itm_source=infoq&itm_medium=popular_widget&itm_campaign=popular_content_list&itm_content=

There is NO free Lunch!

You want to scale to huge request numbers? Forget classic stateful

You want to write Tera/Peta/Exabytes? Maybe give up some consistency

You want thousands of participating machines? Perhaps need to give up some ordering assumptions?

You want billions of customers? Perhaps invent fallback procedures?

You want ultra-fast responses? Learn async communication

When google, myspace, facebook etc. started, they had to go back to FIRST PRINCIPLES and ask hard questions: do you really need xxxxx?

Total end-to-end System Engineering

Front-end technologies

Databases

Networking

Protocols/API Design

Queuing Networks

Operating Systems

Warehouse-scale computing

Queues

Web Server

Cluster Technologies

Monitoring

And much much more….

A single request touches ALL these things. That is the reason DS-problems are so hard to diagnose and fix.

Why Distribute?

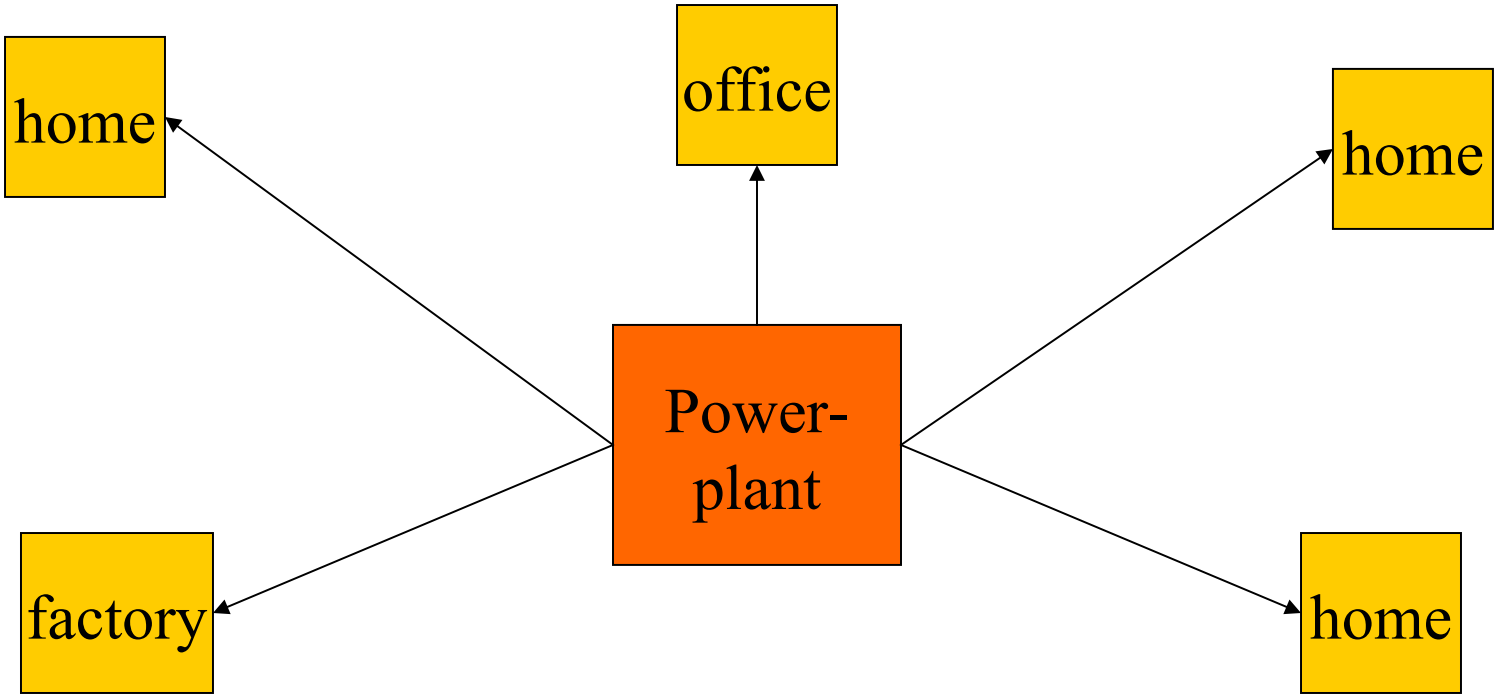
- Robustness/resilience: avoid single points of failures (e.g. Use hot stand-by data centers) with replication
- Performance: Split processing into independent parts
- Scalability/Throughput: allow millions of requests/sec
- Security: create different security domains
- Price per request: use cheaper horizontal scaling or free resources

See: M.Cavage, There's Just No Getting around It: You're Building a Distributed System, ACM Queue, April. 2013

Examples of Distributed Systems

- Energy grid, telcom net
- Villages, towns and big cities
- It-Infrastructure of large companies
- High-performance clusters
- Google, Facebook and Co.
- The Web
- The human body, organizations, states
- A flock of birds

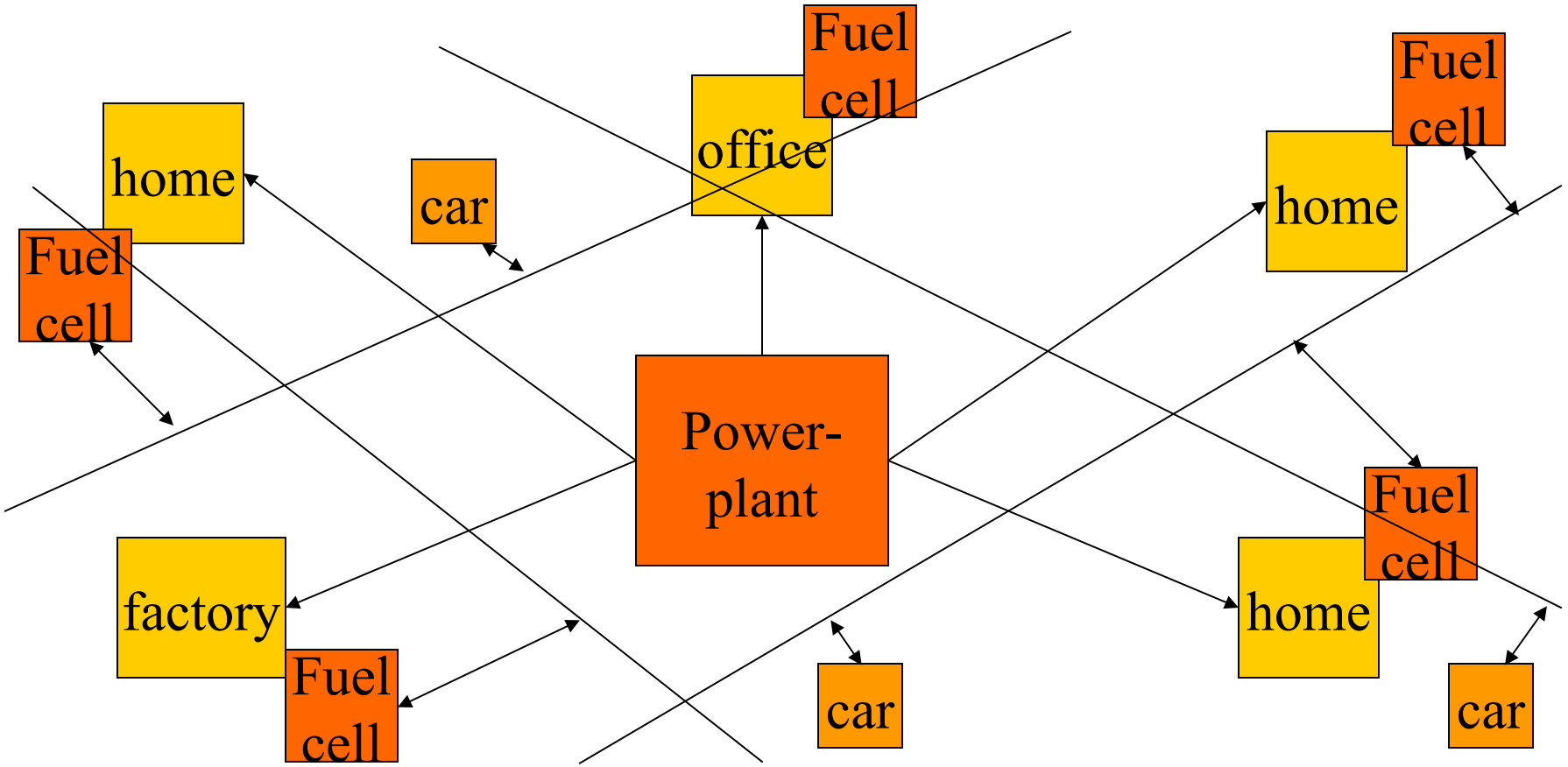
The energy grid now: hub and spoke



Electricity flows in one direction only, with a lot of it lost during transport. Control resides with the power plant.

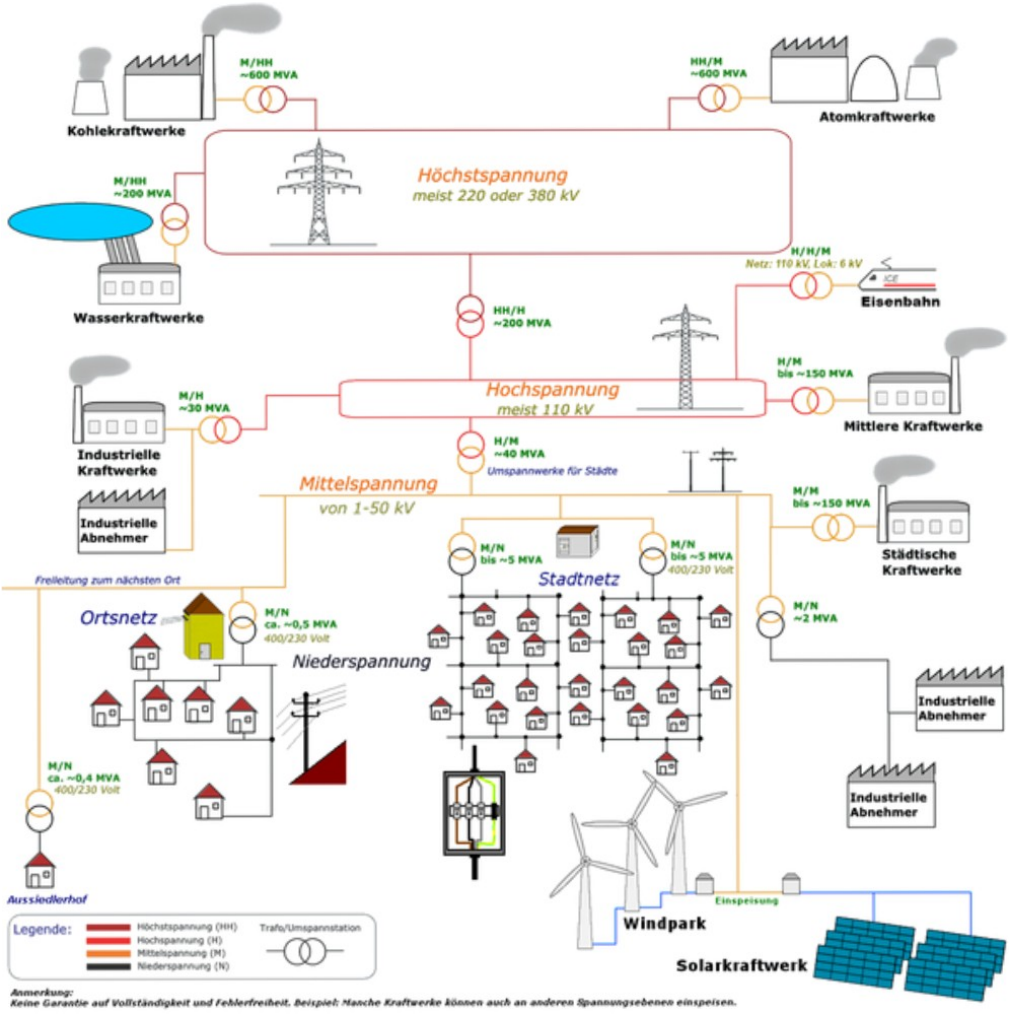
www.wired.com/wired/archive/9.07/juice.html

The future grid : micropower



Power flows many directions, controlled by independent sensors in the grid. A tenfold increase of transactions. Modern GRID computing allows users to tap into a wealth of distributed computing resources. http://www.thegridreport.com/

The German “Energiewende”



Diag: Stefan Riepl, CC Attrib. Share-Alike 2.0, wikimedia commons

Bi-directional flow of energy. Real-time flow control.
Resilient architecture? Load-Balancing, Observability?

De-Carbonization as a DS Problem

STATUS QUO

Current grid management technology: from the 90's

"Manage it? I can't even see it!"

What's included:

- Meters
- Circuits
- Transformers

What's not:

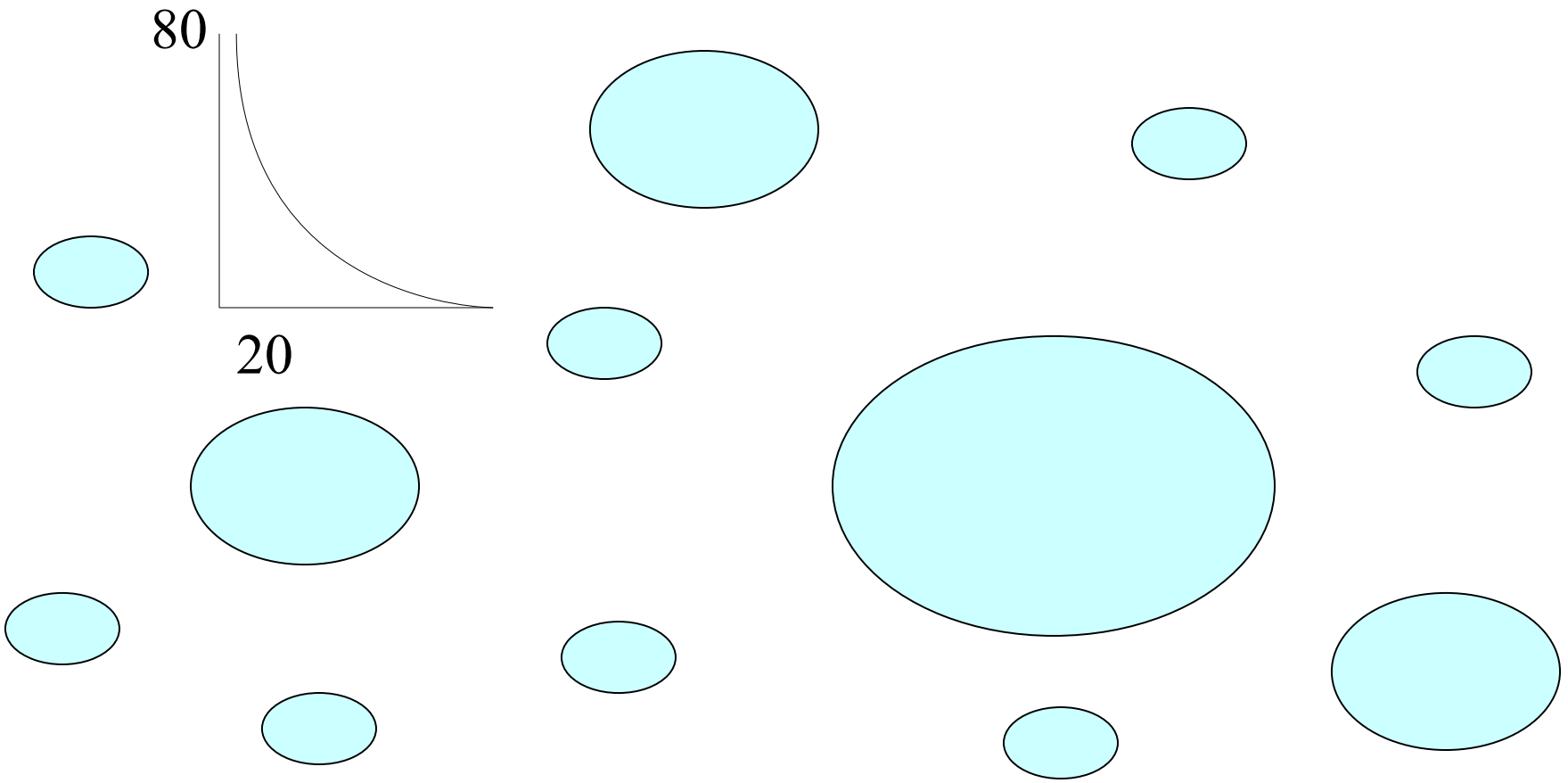
- Flexible loads
- Distributed generation
- Supply & demand

UI from a leading ADMS

Astrid Atkinson, 2022

https://www.infoq.com/presentations/distributed-system-decarbonize/

Scale and Distributions: Power Laws



Villages, Web-Sites, Earthquakes and many others follow power-laws of scale.

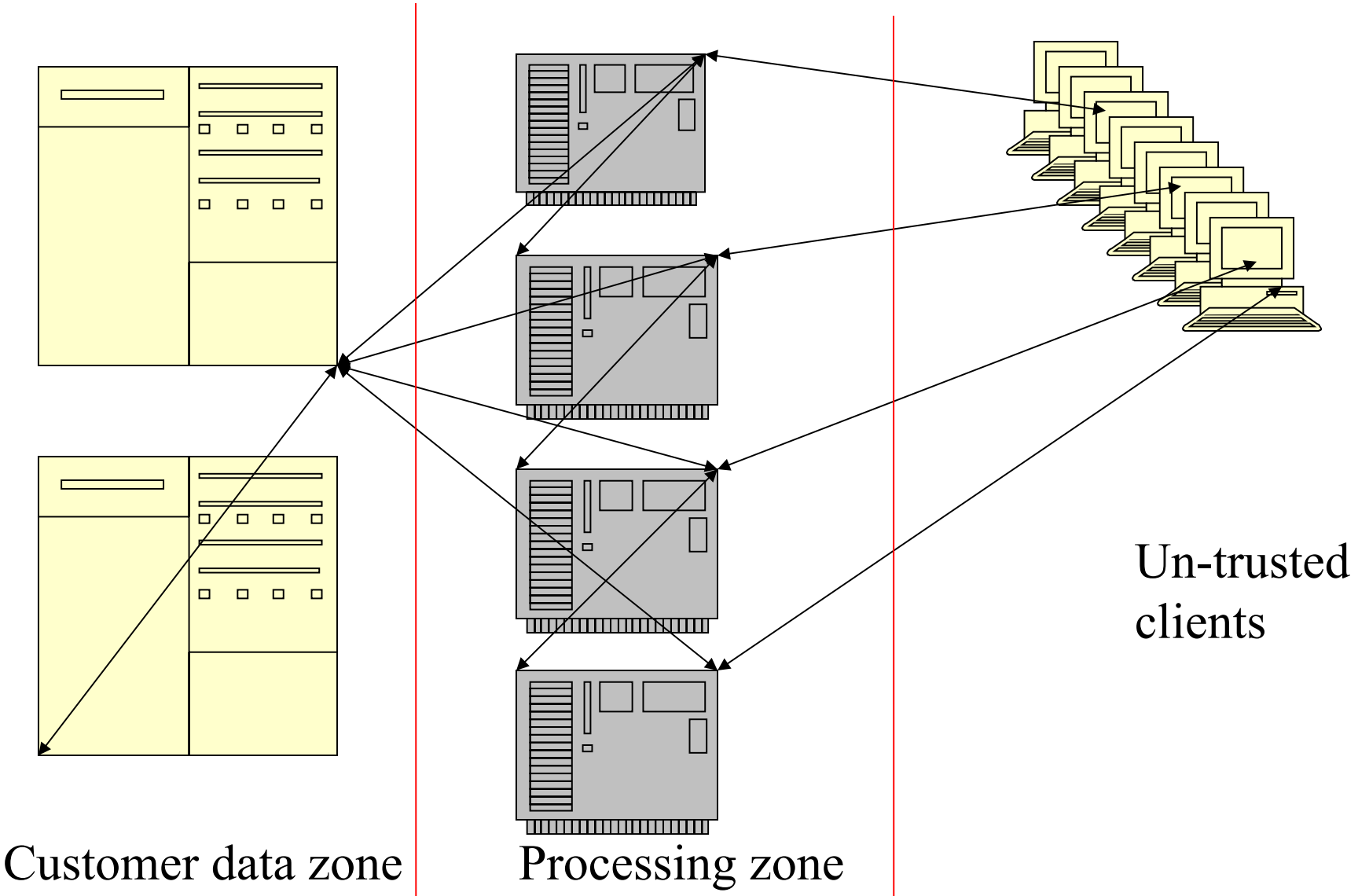
most detailed image of a human cell to date

Found via:
highscalability.com

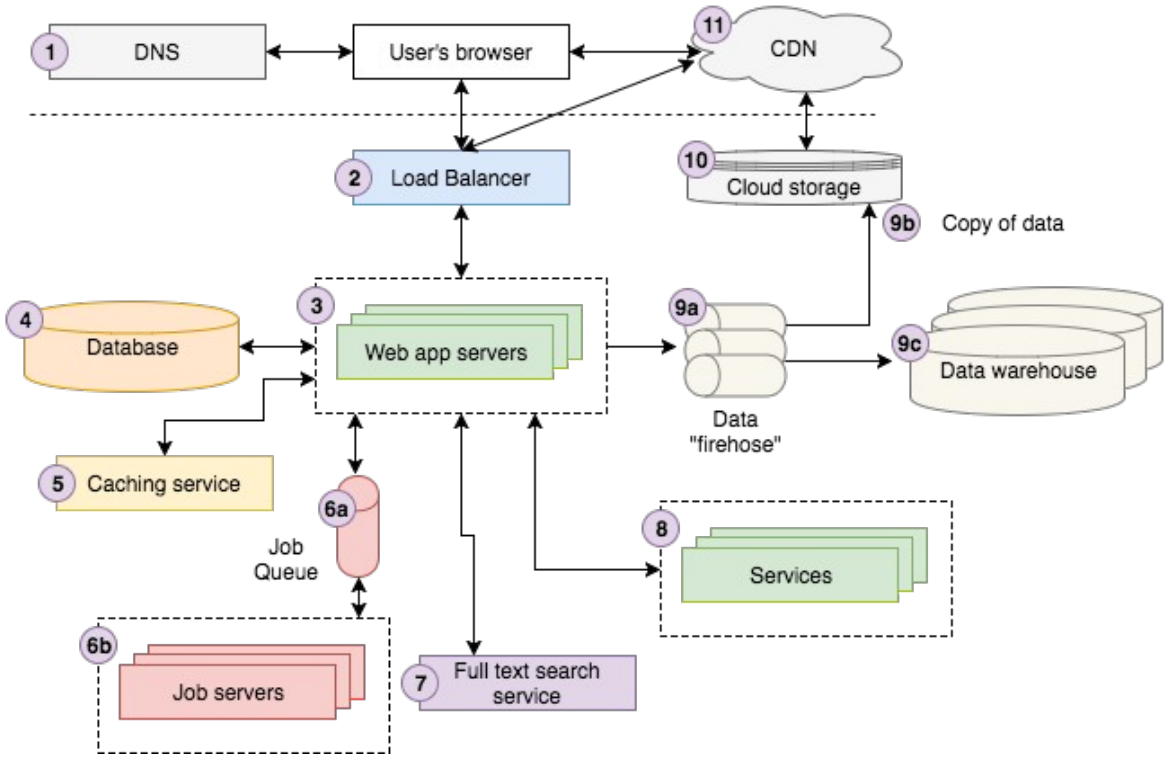
@microscopicture

https://twitter.com/microscopicture/status/1520859755547398149

IT-Infrastructure of large corporations

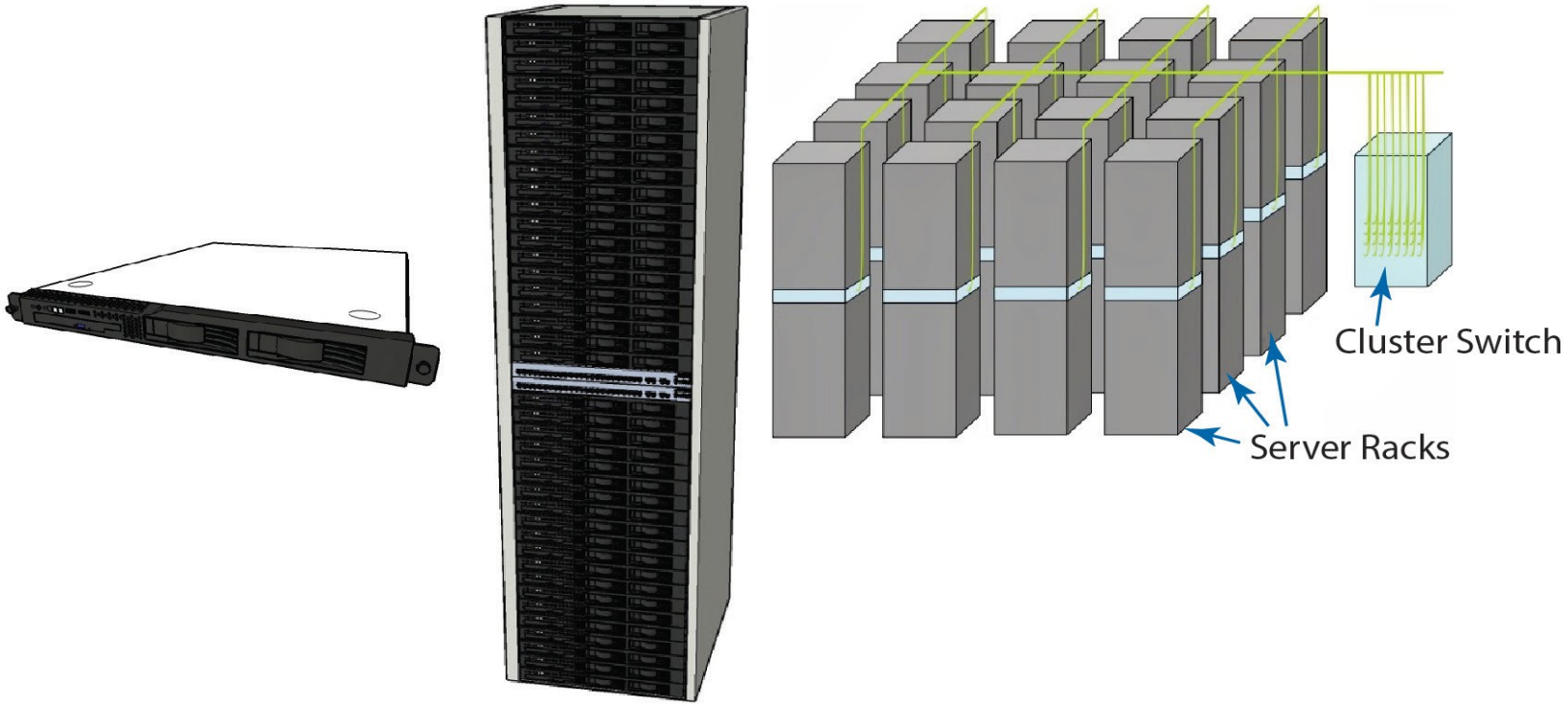


Modern Web Application Architecture



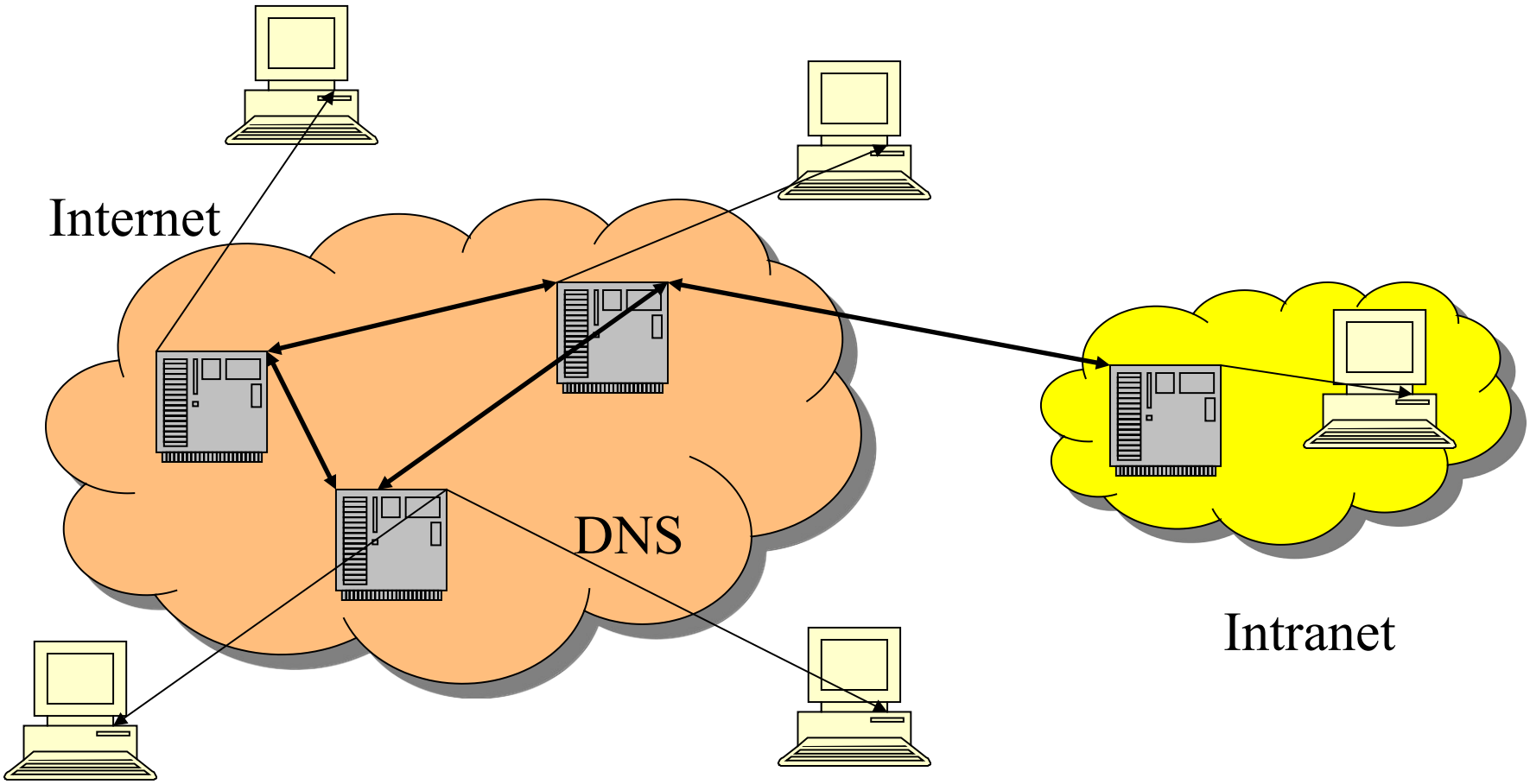
Jonathan Fulton, Storyblocks, Web Architecture 101
The basic architecture concepts I wish I knew when I was getting started as a web developer

Warehouse-Scale Machines



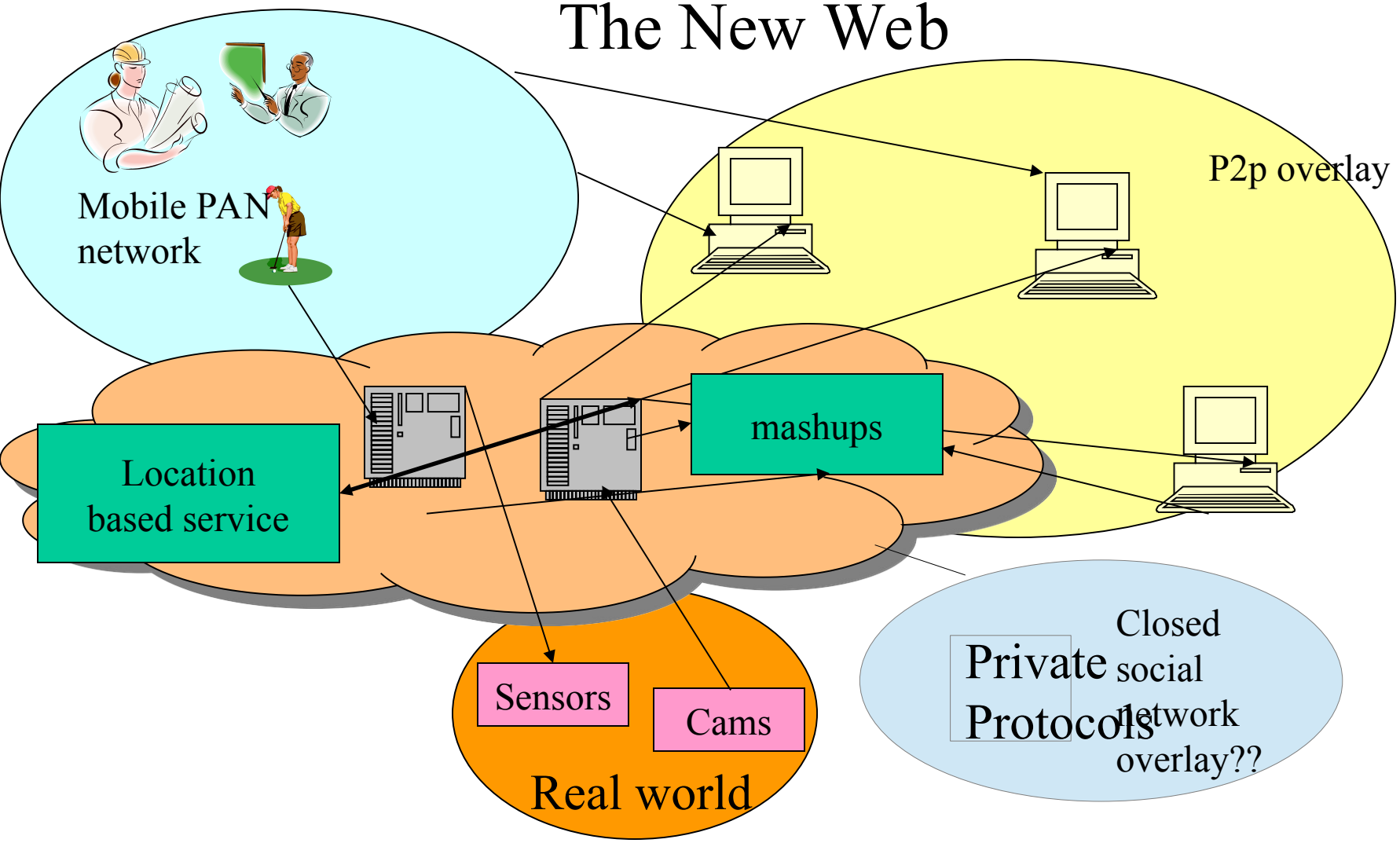
Datacenters are huge distributed systems, requiring special middleware. Diagram from: Barroso et.al., The Datacenter as a Computer. Jupiter network bandwidth 2015: 1.5 Pb/sec. (reads US-library of congress in 1/10 sec.)

World Wide Web



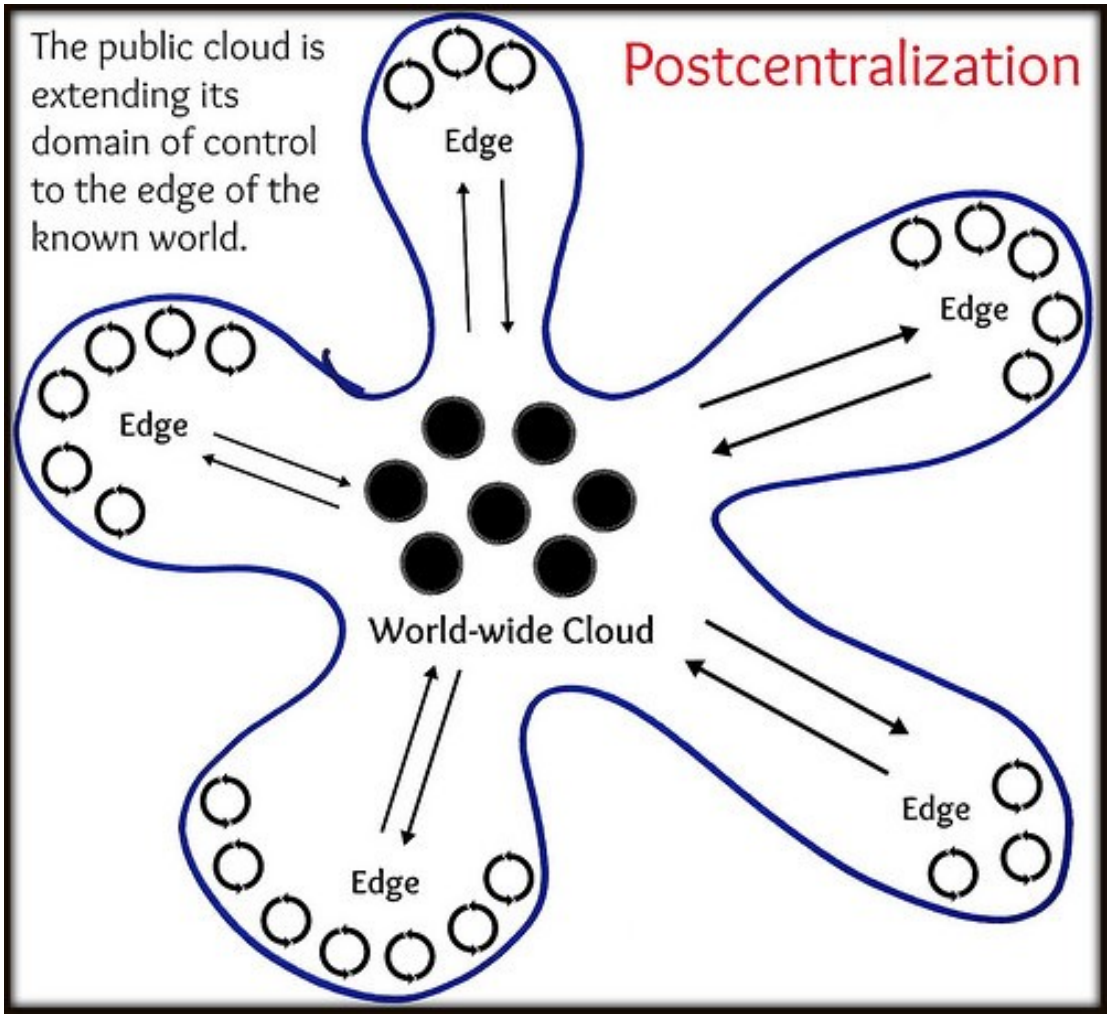
Dialup clients live at the „edges“ of the internet (no fixed IP address, slow upload). How many graphs are layered on top of the physical network structure? (hyperlinks, search-engines, DNS)

The New Web



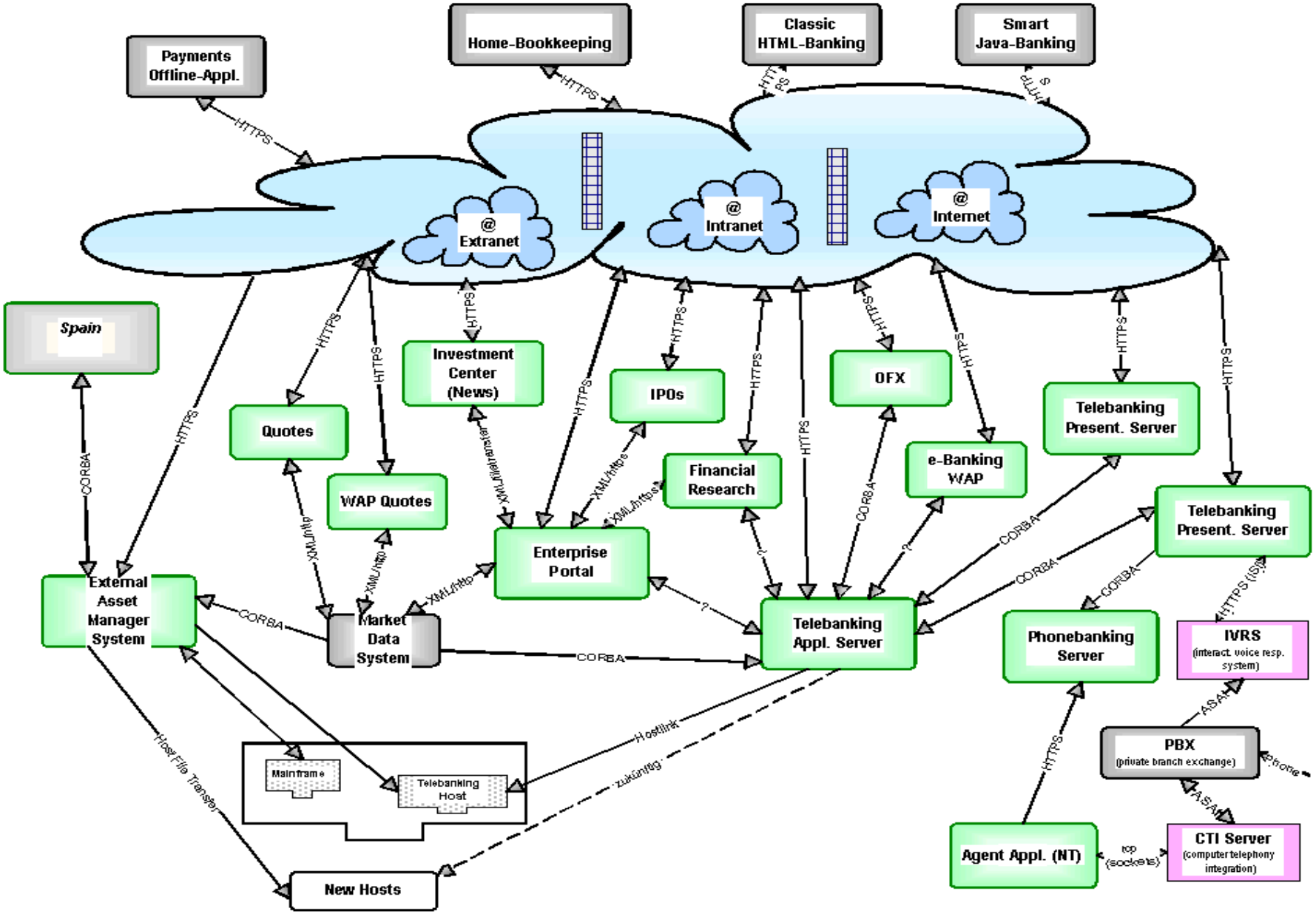
Aggregation of external information and collaboration based on social networks will bring new forms of content production and consumption and consumer areas will influence companies („consumerization“, Gartner Group). More interconnection of different net-types brings more emergent phenomenons.

Cloud and the Edge



Todd Hoff,
Public Cloud Postcentralization is the Thin Edge of the Wedge into the Enterprise

Distributed Application Structure

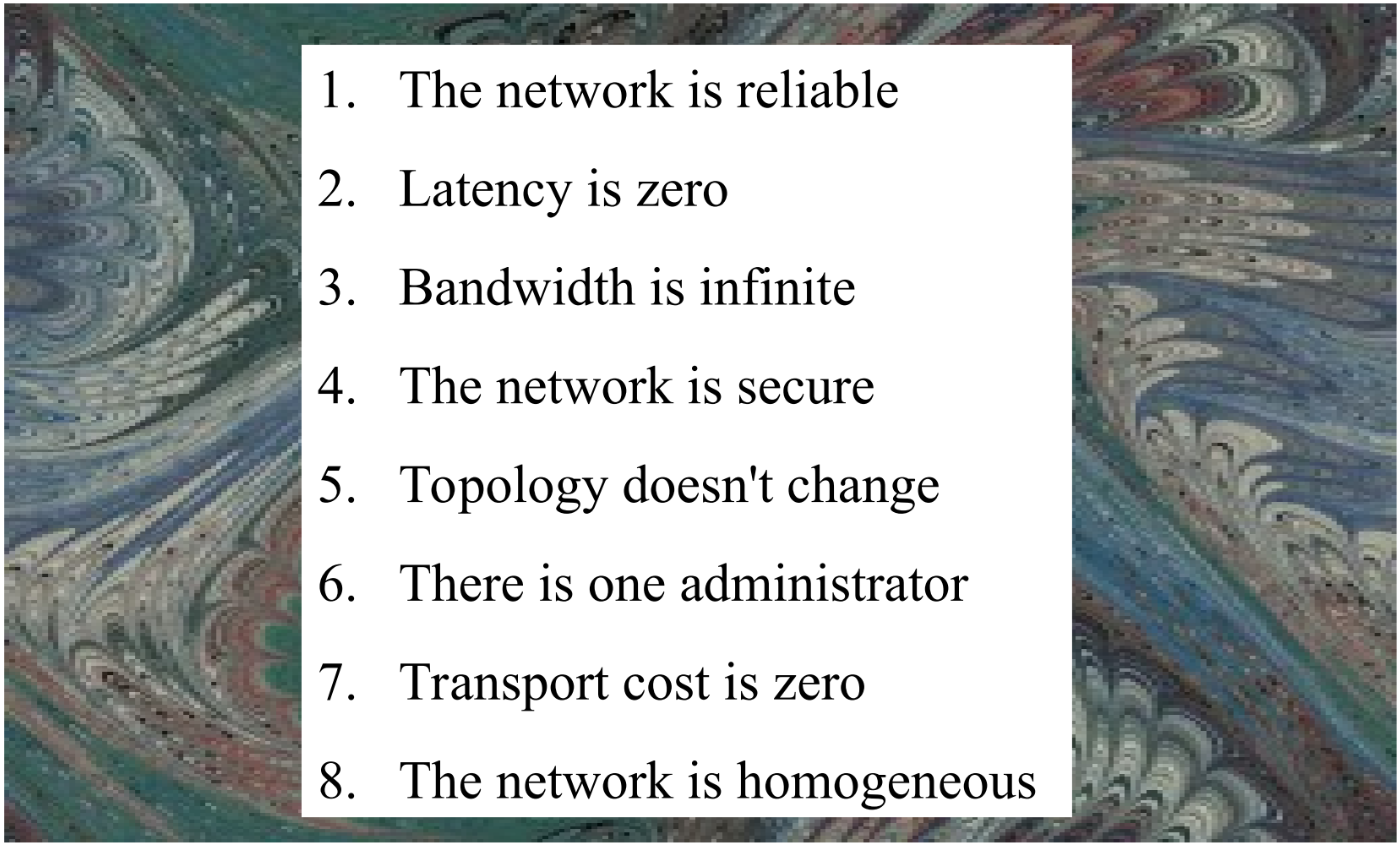


Characteristics of Distributed Systems

- influence of distribution topology and remoteness
- emergent behaviors, concurrent events
- few analytic solutions, few model-based approaches
- heterogeneous components
- no global time
- a strong need for security
- concurrency, parallelism and replication
- failure models define everything!

Don‘t worry, we‘ll dig into all this another day!

The Eight Fallacies of Distributed Computing



“Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences.” (there was an excellent talk at Strangeloop 2014)

Programming Languages and Distributed Systems

Transparency Camp:

-Hide remoteness from programmer

-Create Type-safe interfaces and calls

-Hide Security, Storage and Transactions behind frameworks (.net, EJBs)

-Think DS as a programming model

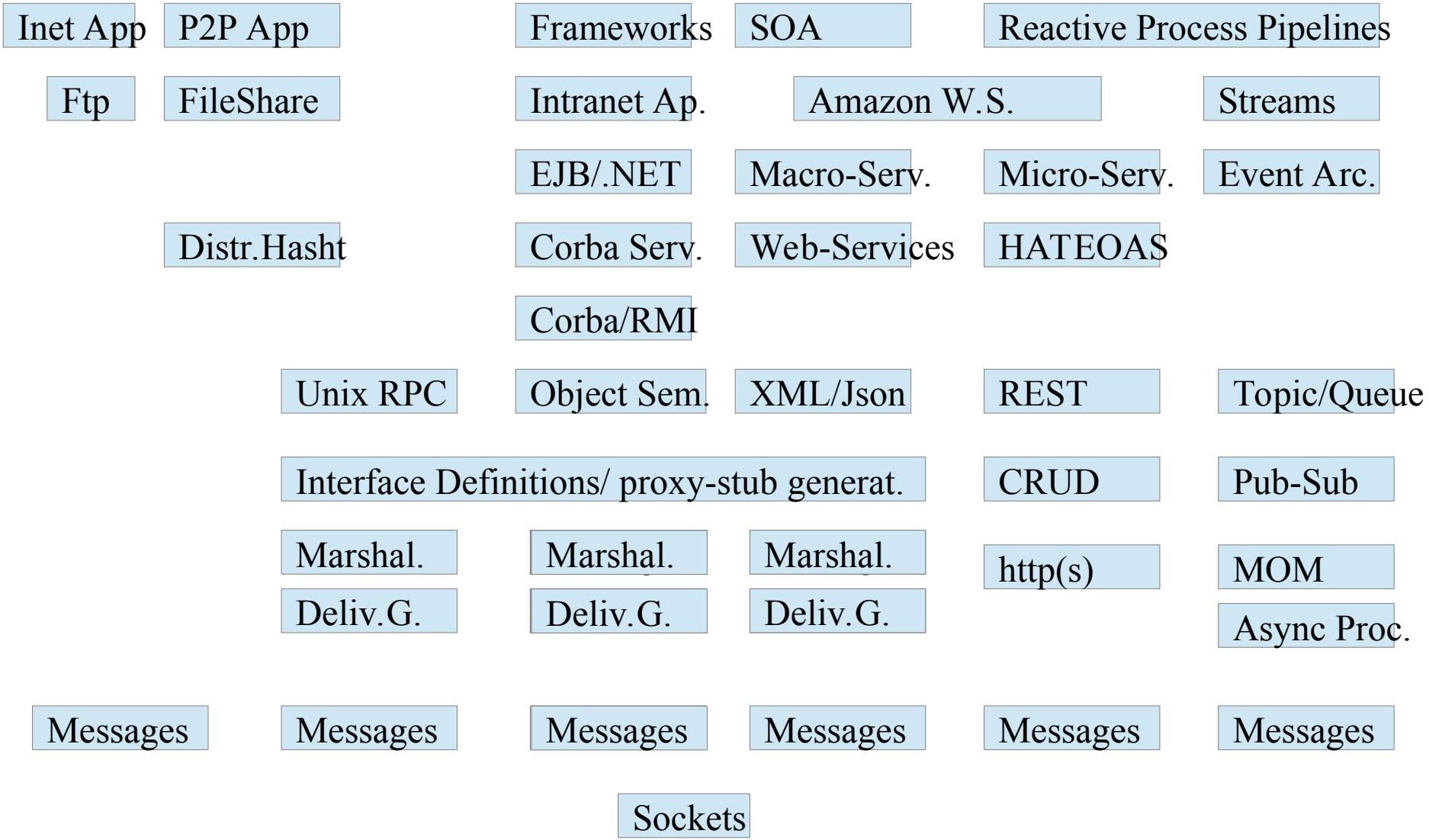
Message Camp:

- simple CRUD interface. Message content is interface
- Coarse grained messages (documents)
- programmers deal with remoteness directly
- Event based or REST architectures

A short history of DS (1)

50s-80s	90s	2000s	>2010
- basic papers on time, consensus, computability etc.	- connecting Intranet applications (CORBA,RPC, COM, DSOM) - C/S WebServer - Programming Models dominate	- peer-to-peer software (file sharing) - large social sites emerge (write probl.) - scalability and performance key - message passing - parallel batch processing (map/reduce) - Ram replaces disc - CAP: AP/CP eventual consistency	- ware-house scale - fan out architectues - online and one-pass algorithms - realtime stream processing - Flash memory - network performance key - microservices and Serverless computing

A short history of DS (2)



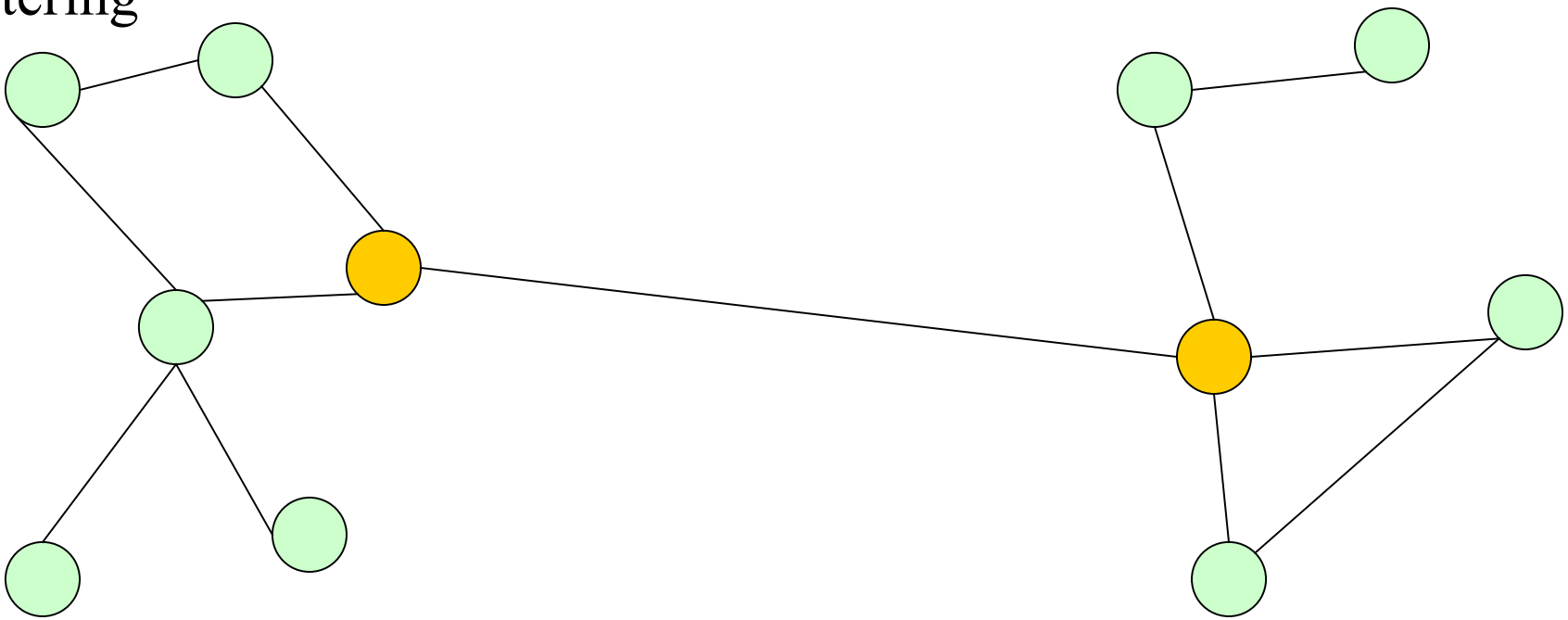
Distribution Topology
e.g.The „small world“ effect:

It takes only a small number of intermediate persons to connect any person on this world to any other one. (A knows B, B knows C, ... F knows G.)

From: The Milgram experiments on social networks. (Andy Oram, Peer-To-Peer, Harnessing the power of disruptive technologies). OpenBC or LinkedIn create a social network from distributed participants.

Topology Effects

High local clustering



How efficient can this DS transport messages? Queries?

How robust is it against random attacks on nodes, targeted attacks on the important connecting nodes?

Metcalfe’s law - Network Effects

- The usefulness of a network grows by the square of the number of users (think about a fax machine – how useful is one?)
- The adoption rate of a network increases in proportion to the utility provided by the network. (That‘s why companies give away software e.g.)
- Are network effects responsible for scale-free (power law) distributions?

Why don't we see lots of facebooks, googles etc.?

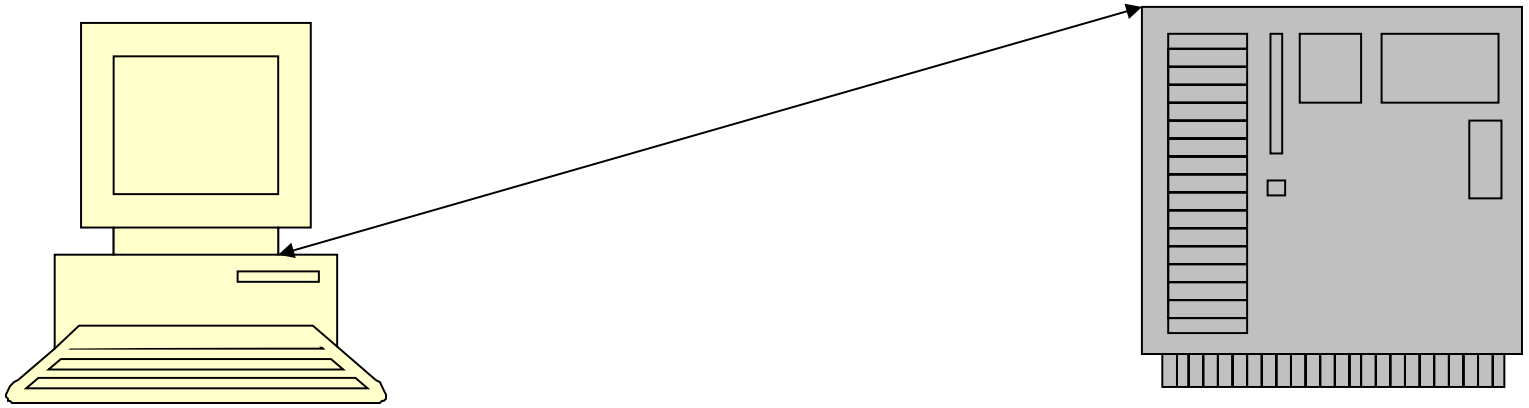
Emergence

„An emergent property is a characteristic of a system that results from the interaction of its components, not from their properties. [] Emergent properties are surprising: it is hard to predict the behavior of the system, even if we know all the rules“. A.B. Downey, Think Complexity (www.thinkcomplexity.com)

Emergent Behavior – a flock of birds

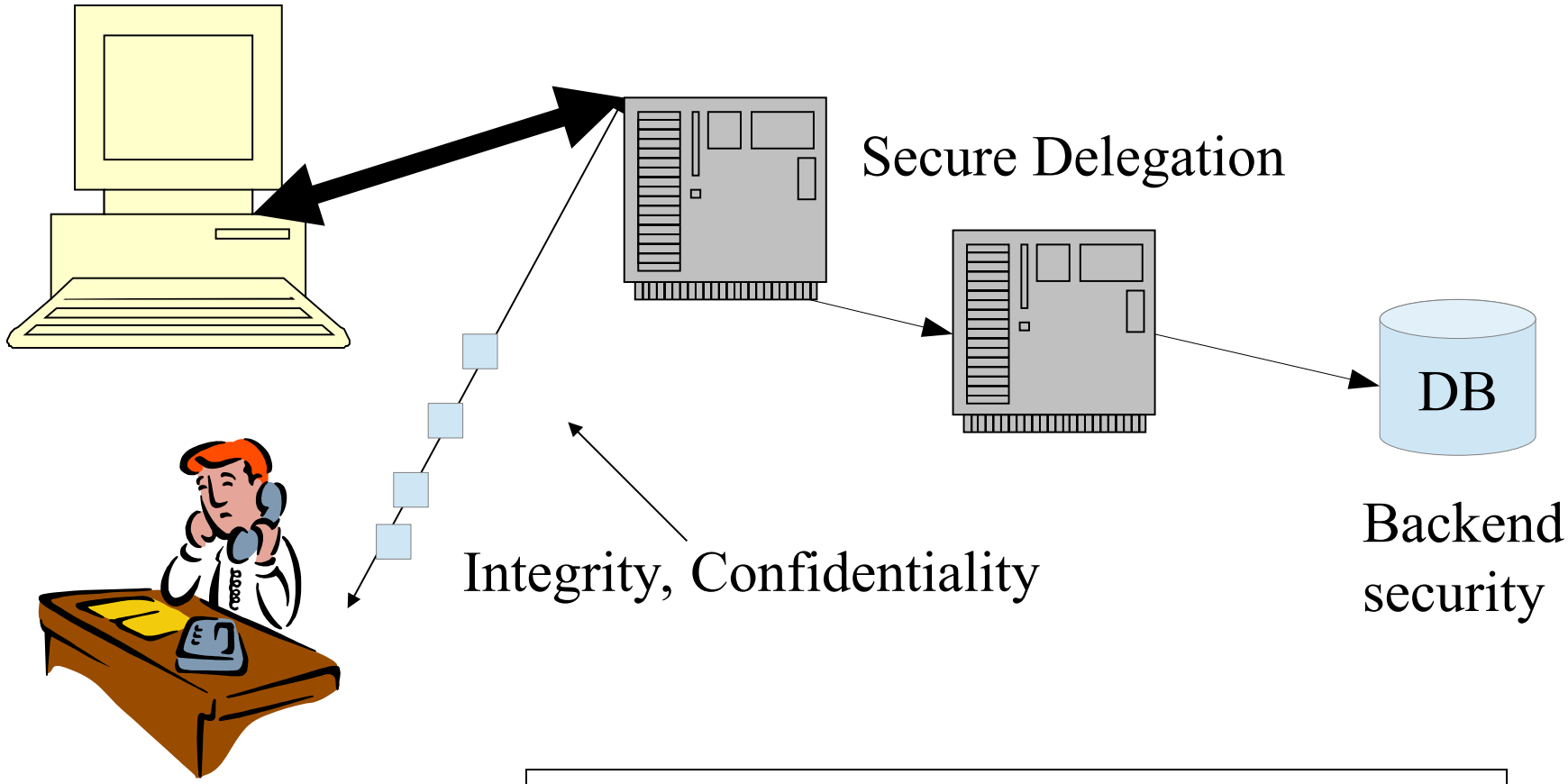
There is no central controller, no „Super-bird“. No bird has a representation of the figure in its head. Instead, every bird follows very simple rules. The resulting figure shows EMERGENT behavior. Many distributed systems show it as well – for good or for bad. (Kevin Kelly, Out of Control – The biology of the new machines. Peter Wegner, Interaction vs Algorithm.) Picture: http://www.pdfnet.dk/ (PD)

Heterogeneous Components



Hardware unreliable	Fault tolerant hardware
Frequent downtimes	System management
Little endian byte order	Big endian byte order
Java Data Types	C++ data types
No callbacks	Fast, access controlled
Slow, no access control	

Security in Distributed Systems



But: sometimes anonymity is needed!!

(peer-to-peer systems)

Authentication

Authorization

Security Topics

- Authentication (who are you?)
- Authorization (what can you do?)
- Confidentiality (can someone spy on us?)
- Integrity (Did somebody change your message?)
- Non-repudiation (It was you who ordered X)
- Privacy/Anonymity
- Firewalls
- Certificates, Public Key Infrastructure, Digital Signature
- Encryption (methods and devices)
- Software Architecture
- Intrusion Detection
- Sniffing
- PGP, SSL etc.
- Denial of Service attacks

Theoretical Foundations of DS

- No global time (logical clocks, vector clocks)
- FLP theorem of asynchronous systems
- The problem of failure detection and timeout
- Concurrency and deadlocks
- CAP theorem: consistency, availabilty and partitioning: choose two only!
- End-to-end argument
- Consensus, leader selection, etc.

Important Programming Terms for DS

- Identity
- Value vs. Reference
- Exception
- Interface vs. Implementation
- Interface Definition Languages (IDL)
- Quality of Service (QOS)
- Stubs/Proxies

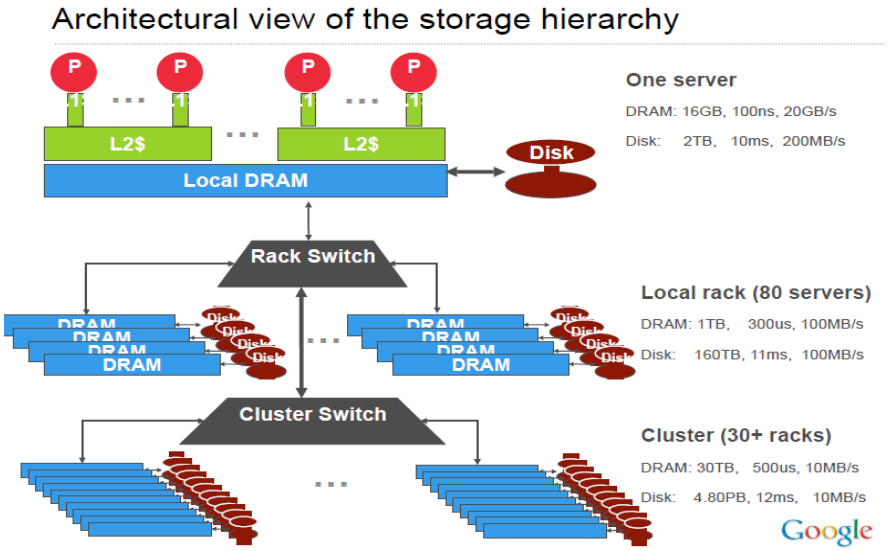
Distributed System Design

- Common Problems (performance, fail-over, maintenance, policies, security integration)
- Information Architecture (define and qualify the information fragments and flows)
- Distribution Architecture (create a map of all participating systems and their quality of service)

Methodologies

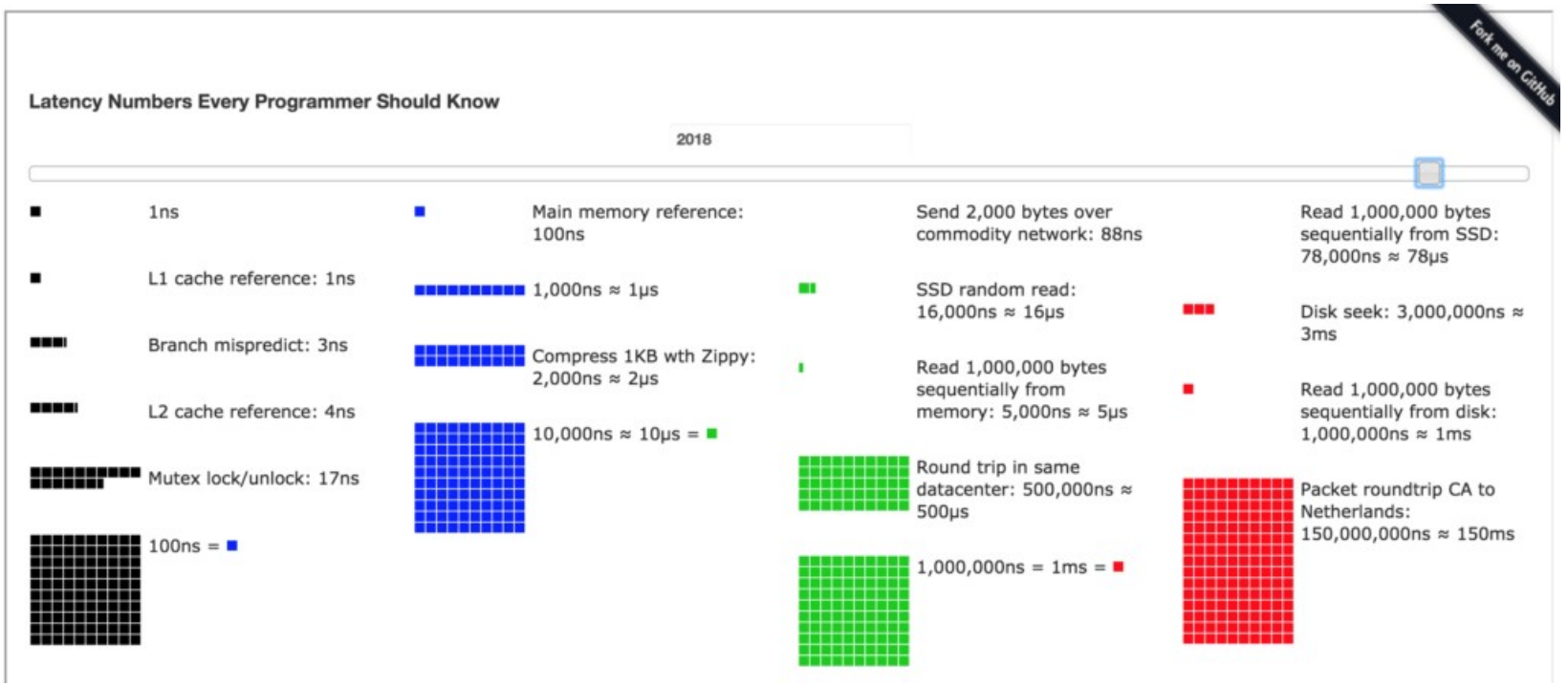
- Analytic Solutions: Queuing Theory based
- Simulation (Spin, Promela, etc.)
- TSA+, Leslie Lamport
- Jeff Dean (Google), Back of the envelope calculations

Know your numbers!
- read/write ratios, random, sequential
- bandwidth, latencies
- hardware (solid state etc.)
- memory hierarchy
- thread-level parallelism and multi-cores



Designs, Lessons and Advice from Building Large Distributed Systems, Jeff Dean, Google

Latency Numbers



https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Advanced Napkin Math: Base Rates

Base Rates: Performance

https://github.com/sirupsen/base-rates, contribute your own

- Sequential Memory Reads <64 bit>: 1 ns @ 6 GiB/s (1MiB: 150 µs, 1 GiB: 150ms)
- Sequential Memory Writes <64 bit>: 5 ns @ 1.5 GiB/s (1MiB: 600 µs, 1 GiB: 600ms)
- Random Memory Read <64 bit>: 25 ns @ 300 MiB/s (1 MiB: 3.5ms, 1 GiB: 3.5s)
- Sequential SSD Read <8 KiB>: 1 µs @ 4 GiB/s (1 MiB: 200 µs, 1 GiB: 200 ms)
- Seqential SSD Write <16 KiB>, No Fsync: 15 µs @ 3.5 MiB/s (1 MiB: 250ms, 1 GiB: 5 min)
- TCP Echo Server, Localhost <64 bytes>: 15 µs
- Random SSD Read <64 bits>: 100 µs @ 0.5 MiB/s (1 MiB: 1.5 s, 1 GiB: 0.5 hour)
- Cloud Within-Zone Roundtrip: 250 µs
- Sequential SSD Write <16 KiB>, Fsync: 5 ms @ 10 KiB/s (1 MiB: 100s, 1GiB: 1 day)

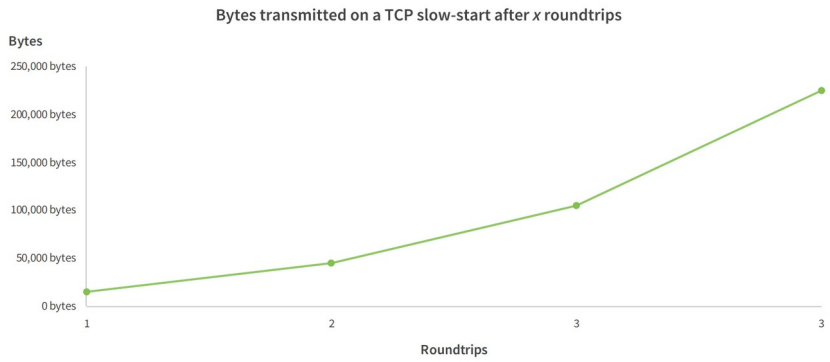
Base Rates: Networking

Rounded numbers from https://wondernetwork.com/pings to make them easier to remember.

- D.C. -> Frankfurt: 100ms
- D.C. -> {Singapore, Sydney}: 250ms
- D.C. -> Los Angeles: 60ms
- D.C. -> Tokyo: 150ms
- D.C. -> Kansas City: 40ms
- Singapore -> {Sydney, Tokyo}: 100ms
- D.C. -> Sao Paulo: 100ms
- Frankfurt -> Cape Town: 150ms

TCP Window-Scaling

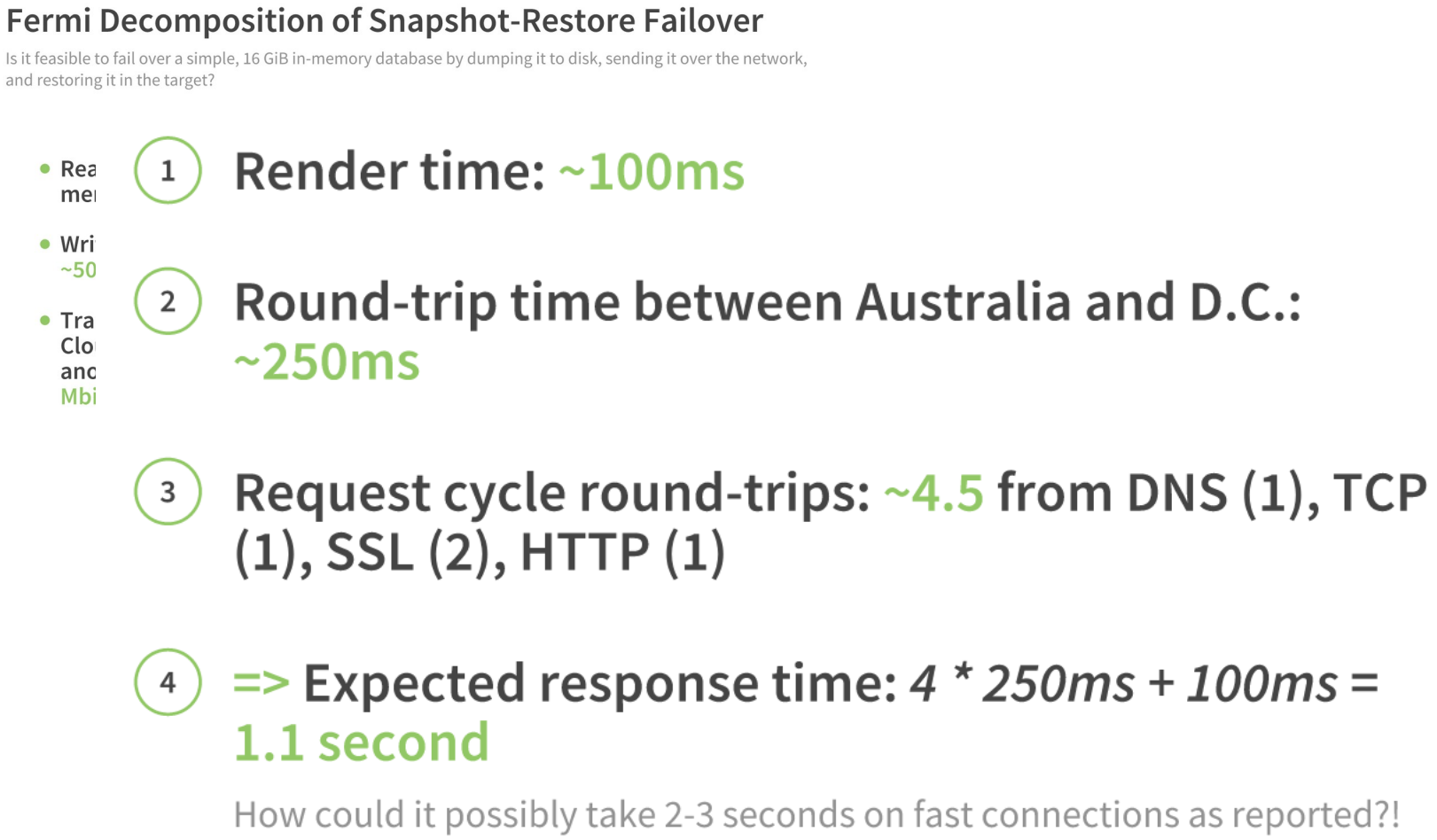
Initial window is 10 * 1,500 bytes, and each roundtrip (if no packets are lost) will double the window size.



Also: watch bandwidth delay rate!!

From: Simon Eskildsen, Advanced Napkin Math: Estimating System Performance from First Principles, Shopify, Talk at SREcon 2019 https://youtu.be/IxkSlnrRFqc

Advanced Napkin Math: Applied



From: Simon Eskildsen, Advanced Napkin Math: Estimating System Performance from First Principles, Shopify, Talk at SREcon 2019 https://youtu.be/IxkSlnrRFqc

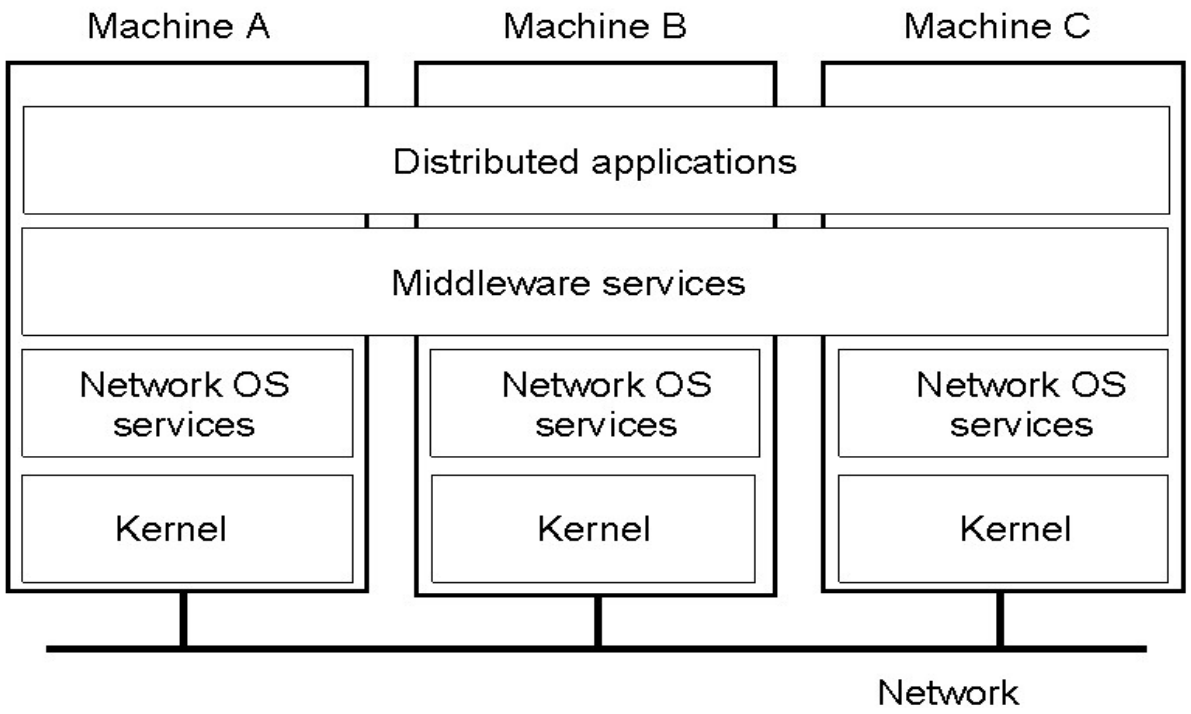
Middleware for Distributed Systems

What is Middleware?

- it is libraries, components or services helping you in creating distributed applications
- it has glue-code and generators to let e.g. c++ clients talk to java servers.
- it controls messages and enforces so called delivery guarantees, e.g. like at-least-once
- it re-orders requests from participants to create a causal or total ordering
- it takes over responsibility for messages, possibly even storing them in between
- it creates groups of nodes which process events together and controls fail-over
- it hides things like differences in hardware, location of services and offers load-balancing
- it allows filtering of requests or provides means to add additional security info to calls
- it does monitoring of requests to the nearest storage subsystem
- it provides powerful services like locking, scheduling and messaging to applications
- powerful frameworks provide automatic storage, security checks and transactional control
- message bus architectures provide loose coupling through pub/sub functions

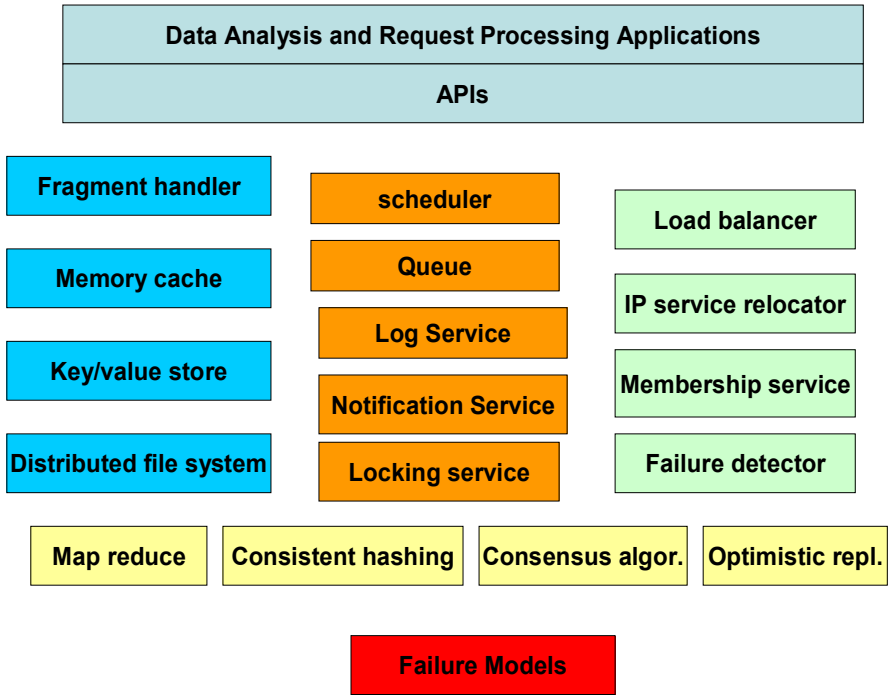
The easiest way to understand the importance of middleware is to start developing a distributed application on the raw socket level and see what is missing. There is no doubt about the need for middleware, just the extent of it is heatedly discussed (e.g. Amazon's Werner Vogel: we don't do frameworks at Amazon..)

General structure of a distributed system as middleware.



From: van Steen/Tanenbaum

Components of a Distributed Operating System



The Transparency Dogma

- Middleware is supposed to hide remoteness and concurrency by hiding distribution behind local programming language constructs

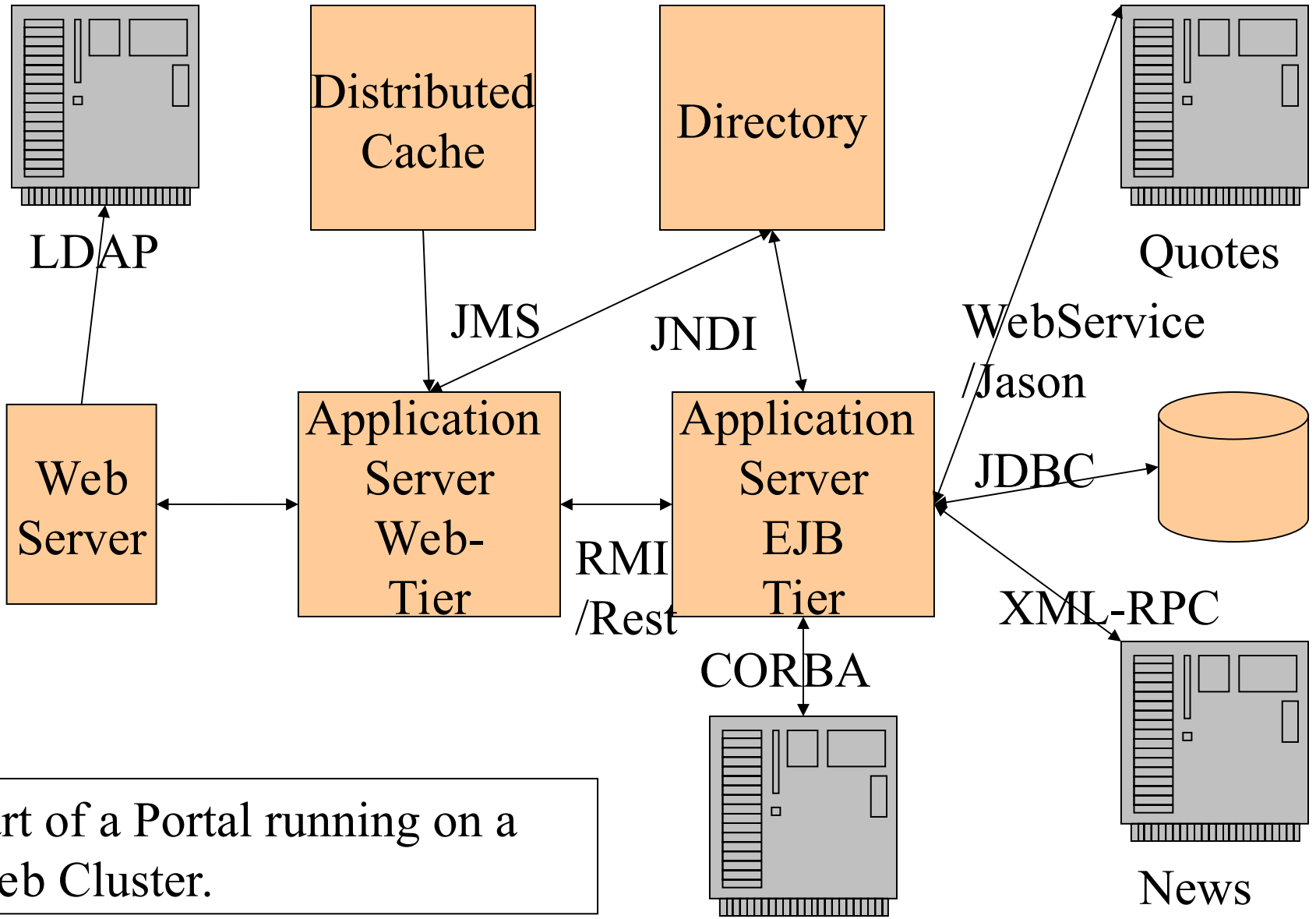
Critique: Jim Waldo, SUN

Full transparency is impossible and the price is too high

Distribution Transparencies

- Access: mask differences in languages and data representation (e.g thrift, protocol buffers)
- Failure: mask failures to enable fault tolerance by automated fail-over to other servers
- Scalability: intelligent load-balancing of requests,
- Redundancy: transparent replication of data
- Location: use logical, not physical names to access services. Allows changing services
- Migration: hide the true location of a service or object from clients. If the location changes, the client won‘t notice it.
- Persistence: automatically load/store data on-demand to unload server memory
- Sharding: distribute storage requests across backend systems (virtual storage)
- Transactions: make requests ACID
- Security: automatically check for the required credentials or roles in requests
- Monitoring: create central logs with correlation IDs joining requests parts across nodes

Where do we find Middleware?



Part of a Portal running on a Web Cluster.

Classification

- Socket Based Services (
- Remote Procedure Calls (RPCs)
- Object Request Brokers (CORBA, RMI)
- Message Oriented Middleware (MOMs) and Event-Driven-Systems, Reactive Systems
- Web-Services (XML-RPC, SOAP,UDDI) and SOA, REST
- Frameworks (Enterprise Java Beans, J2EE)
- Peer-To-Peer (Napster, Gnutella, Freenet, seti@home)
- Agent based (Jini, Aglets)
- Tuple-Spaces, distributed blackboards
- Warehouse-Computing Architectures, Data Centers

RPC type Middleware

- E.g. Sun-RCP, apache thrift, google protocol buffers
- Main idea: allow remote function calls across languages. Provide concurrent and parallel processing of requests
- Has generators to create language specific glue code
- On top of it, all kinds of distributed components can be built: Directory, File system, Security
- Apache Thrift, jnb.ociweb.com/jnb/jnbJun2009.html

Provides the “plumbing layer” of distributed systems: NEVER build remote functions without the help of such a library!

Distributed “Objects”

CORBA

- Object Request Broker
- Multi-language support (platform independence)
- Interface Definition language
- Wire Protocoll: IIOP, GIOP

RMI

- Java only (e.g.Introspection used)
- Lightweight method call semantics
- Java Implementations
- Wire Protocoll now mostly: RMI over IIOP

Both try to preserve object semantics. Interface/Implementation separation. Object semantic not good for the Internet!

Distributed Computing Frameworks

- Objects are too granular: performance and maintenance problems
- Programmers need more help: separation of concerns and context

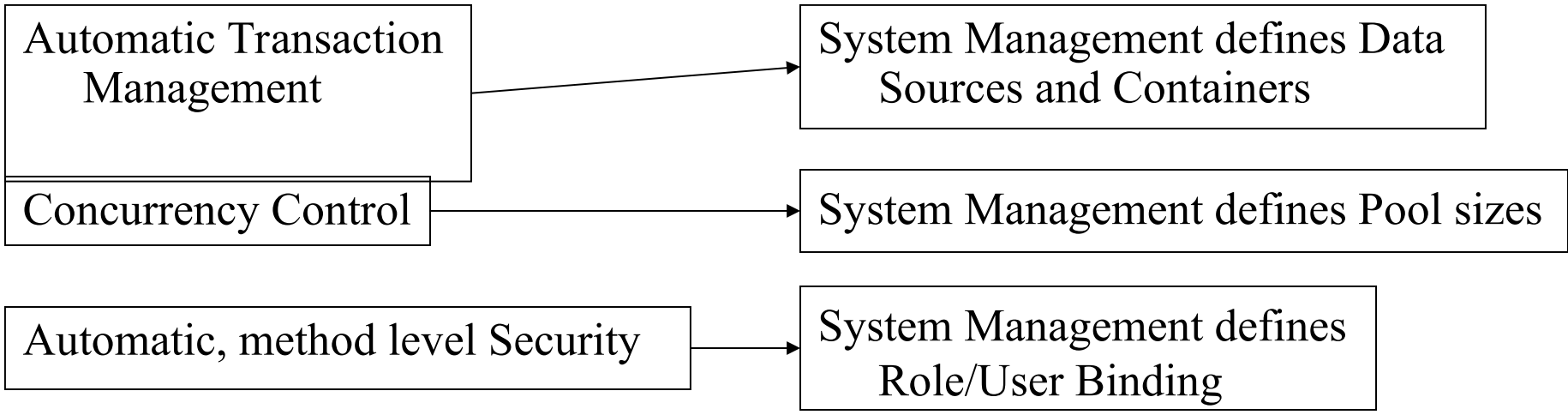
Solutions:

- Enterprise Java Beans
- CORBA Components
- COM+
- .NET
- Spring, Ruby on Rails etc.

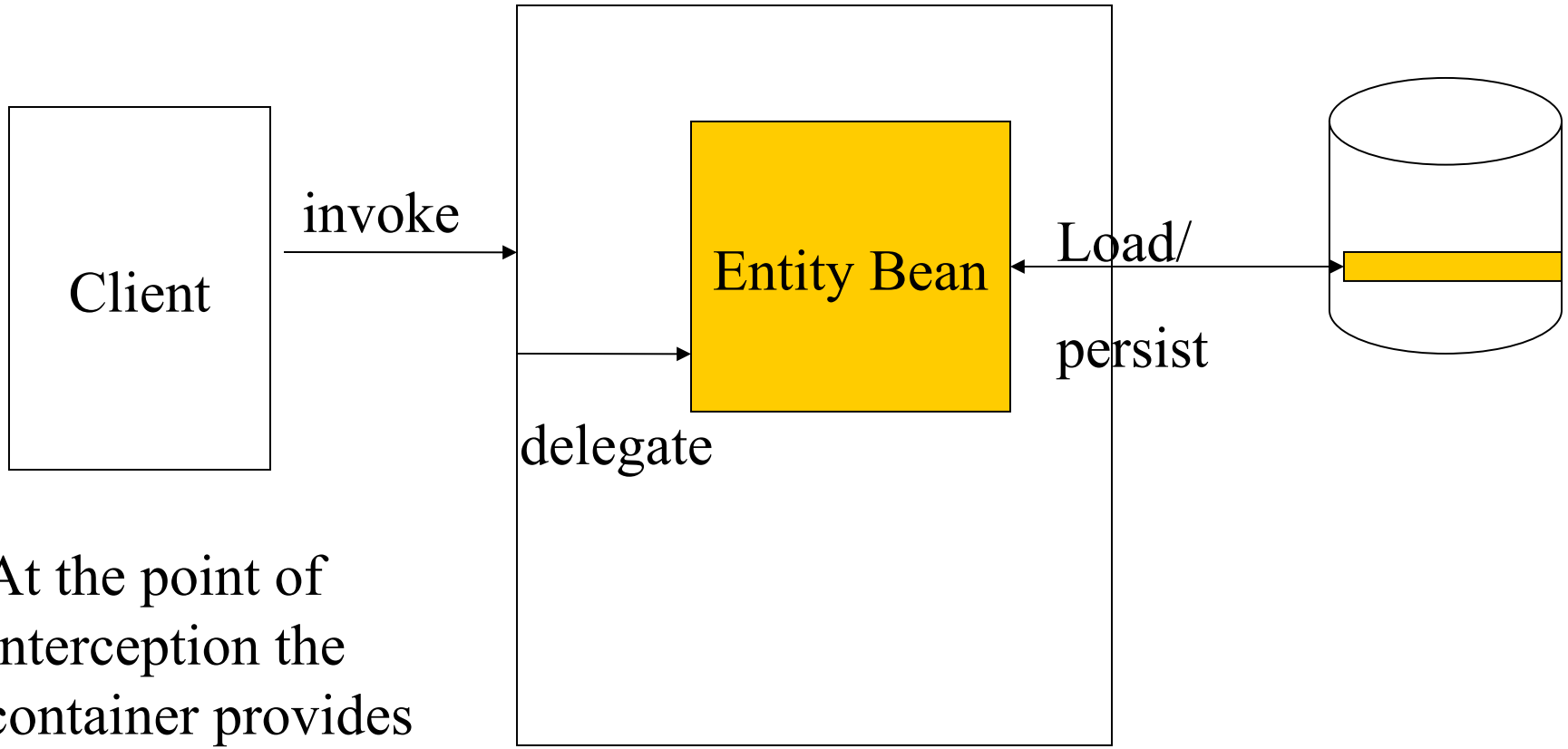
EJB: Concerns and Context

EJB Framework (Separation of **concerns**):

Deployment (Separation of **context**):



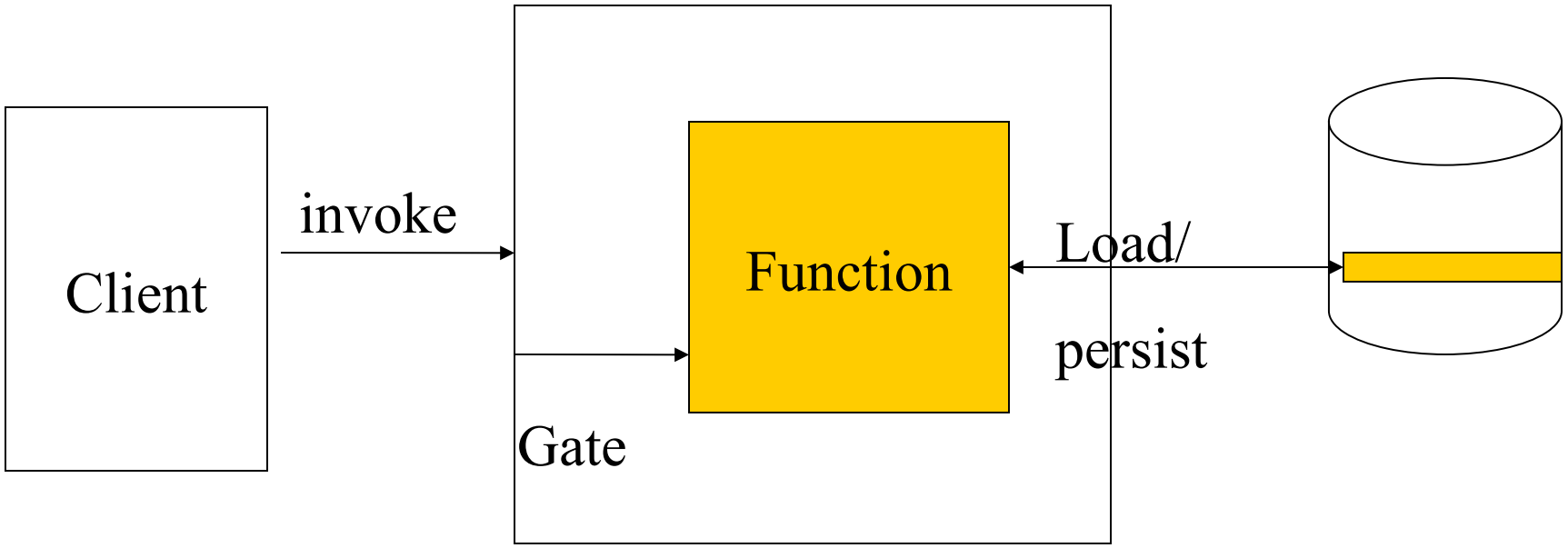
From EJB Container ...



At the point of interception the container provides the following services to the bean:

Resource management, life-cycle, state-management, transactions, security, persistence

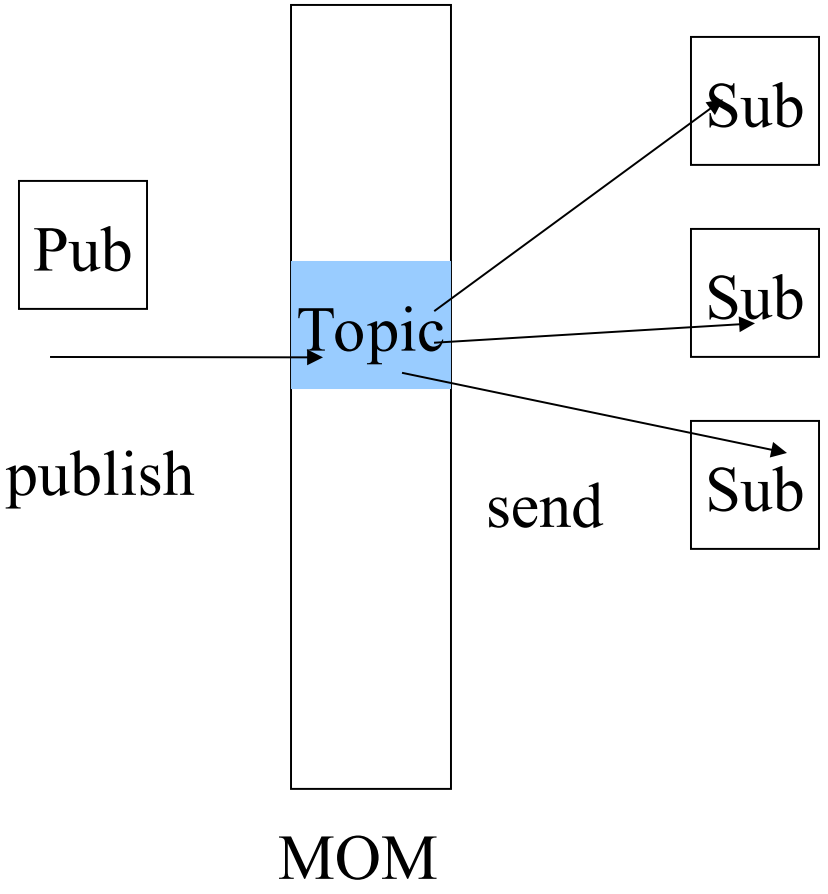
To Serverless Computing

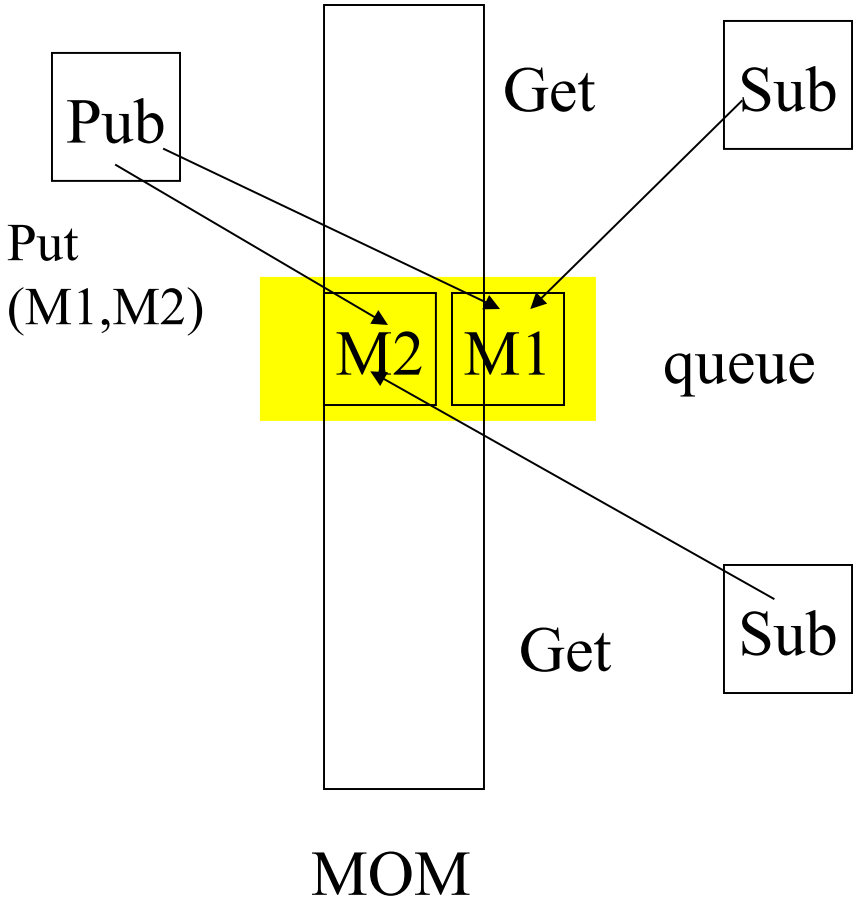


Function as a Service Frameworks provide auto-scaling, life-cycle, security for business logic functions

Distributed Messages (MOM)

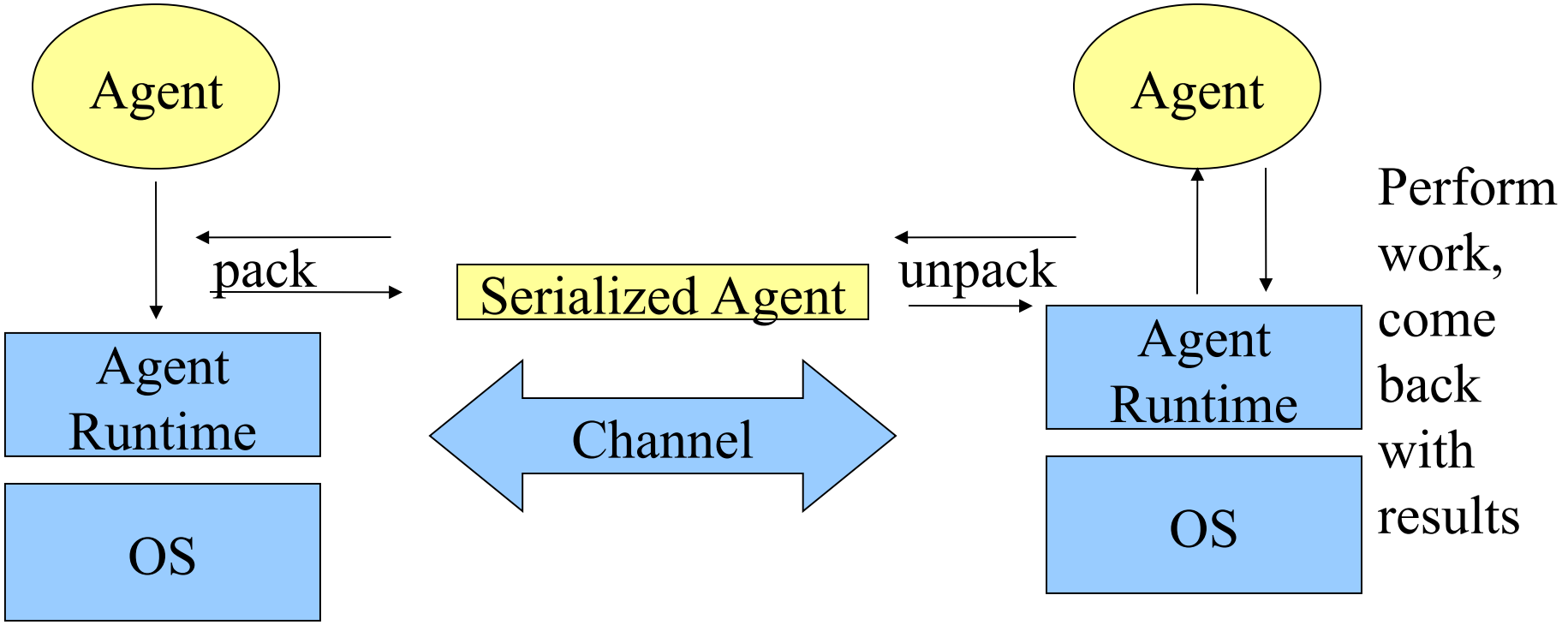
Asynchronous, loosely-coupled (fault tolerant), persistent messages with either publish/subscribe (topics) or queuing semantics. Scales well. Delivery guarantees differ.



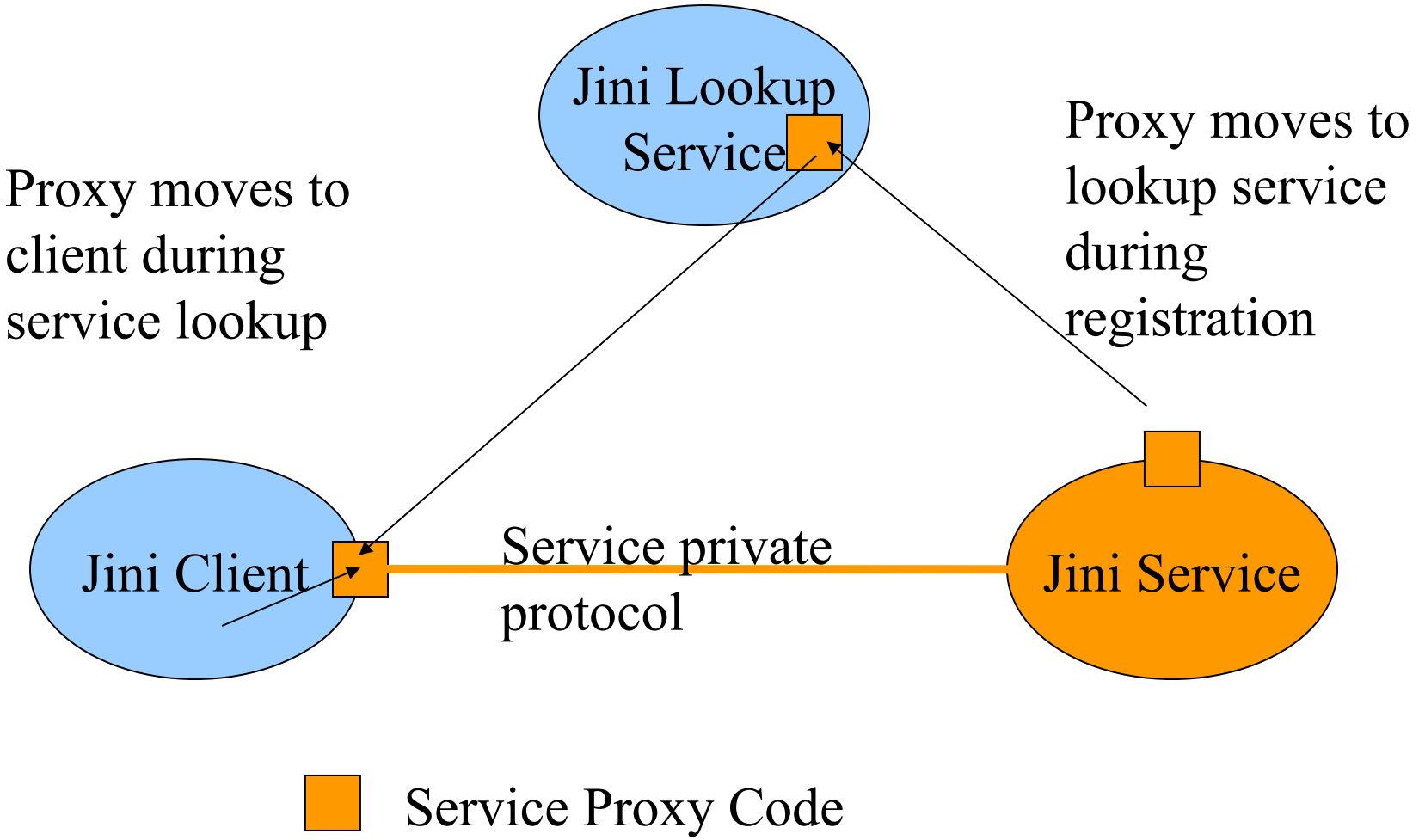


Distributed Code I (Agents, Aglets)

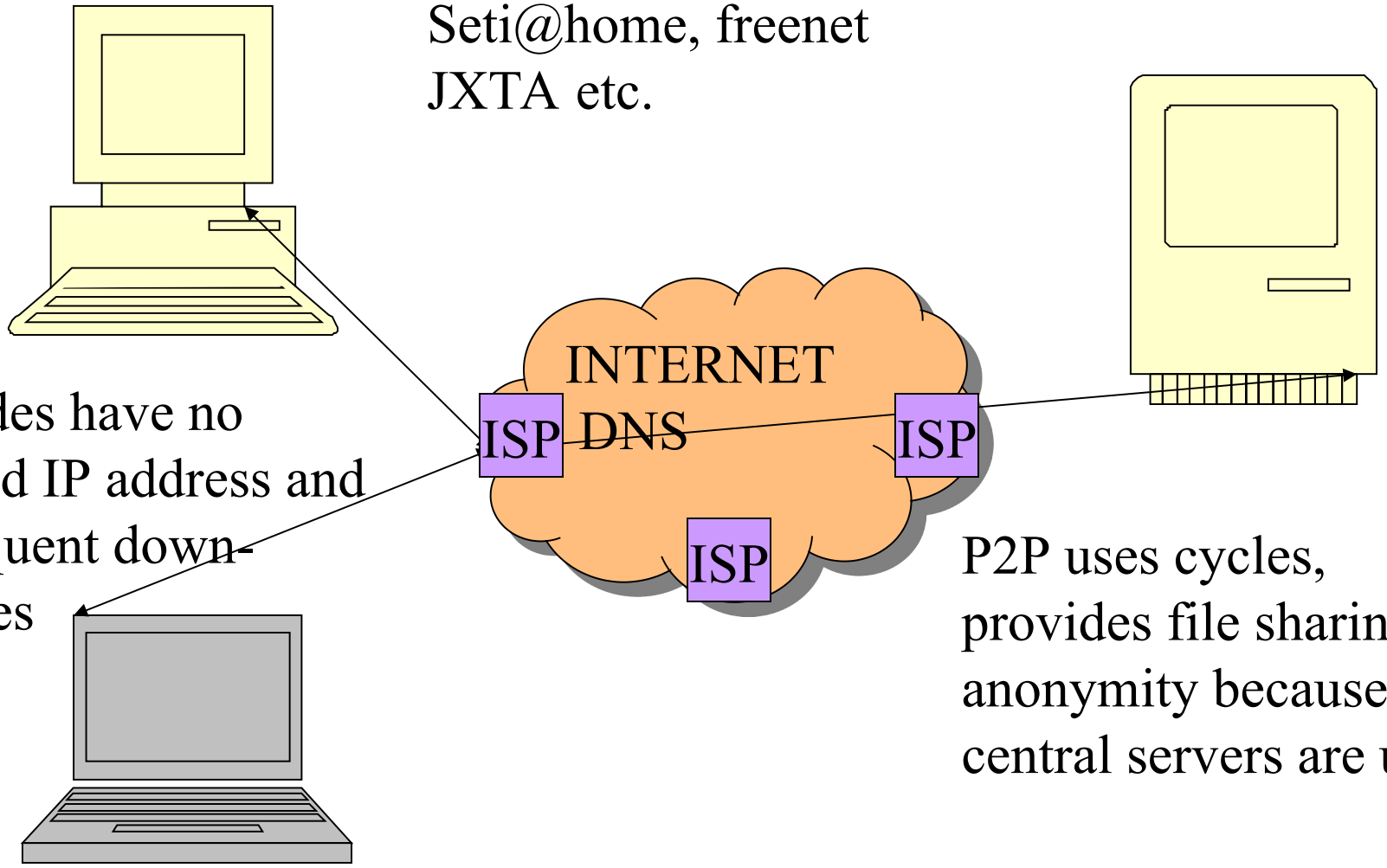
The Problem: who wants a new runtime system?



Distributed Code II (Jini) – The End of Protocols?



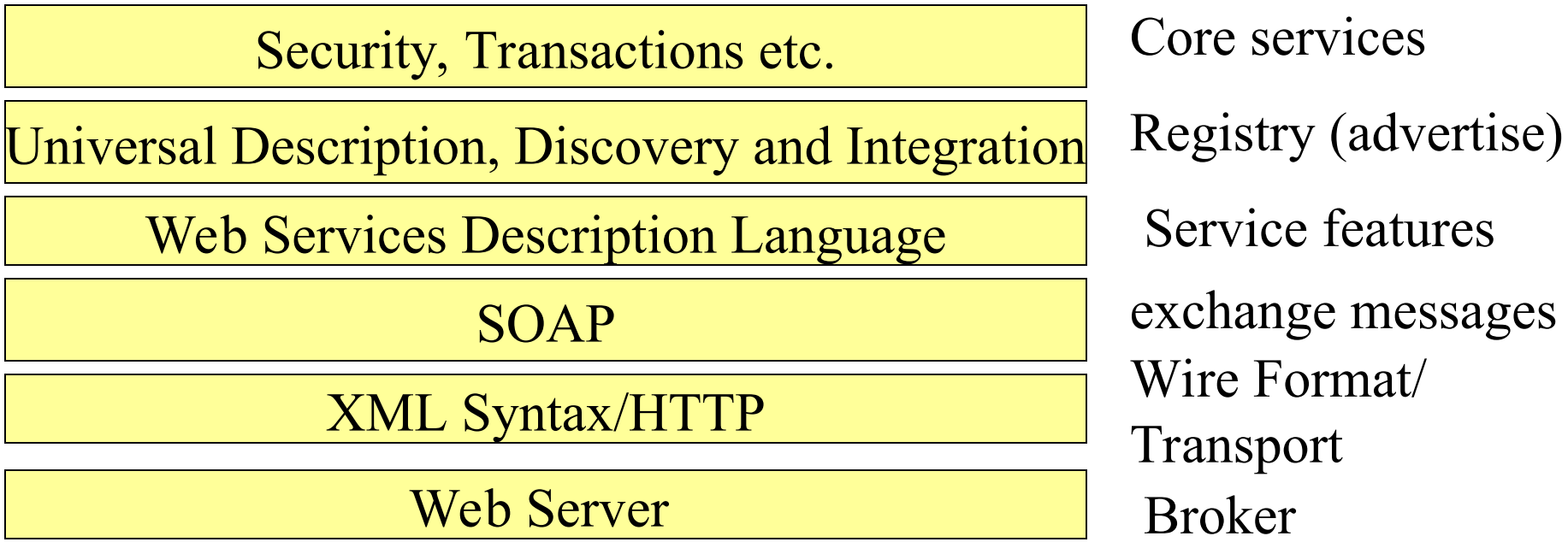
Peer 2 Peer: from shared files to global ledger



Problems: File Versioning? Overhead? Consensus? Security?

WebServices: Same, Same...

Promises de-coupling of service provider and requester, document interfaces, machine-to-machine communication and ease of use compared to distributed objects.



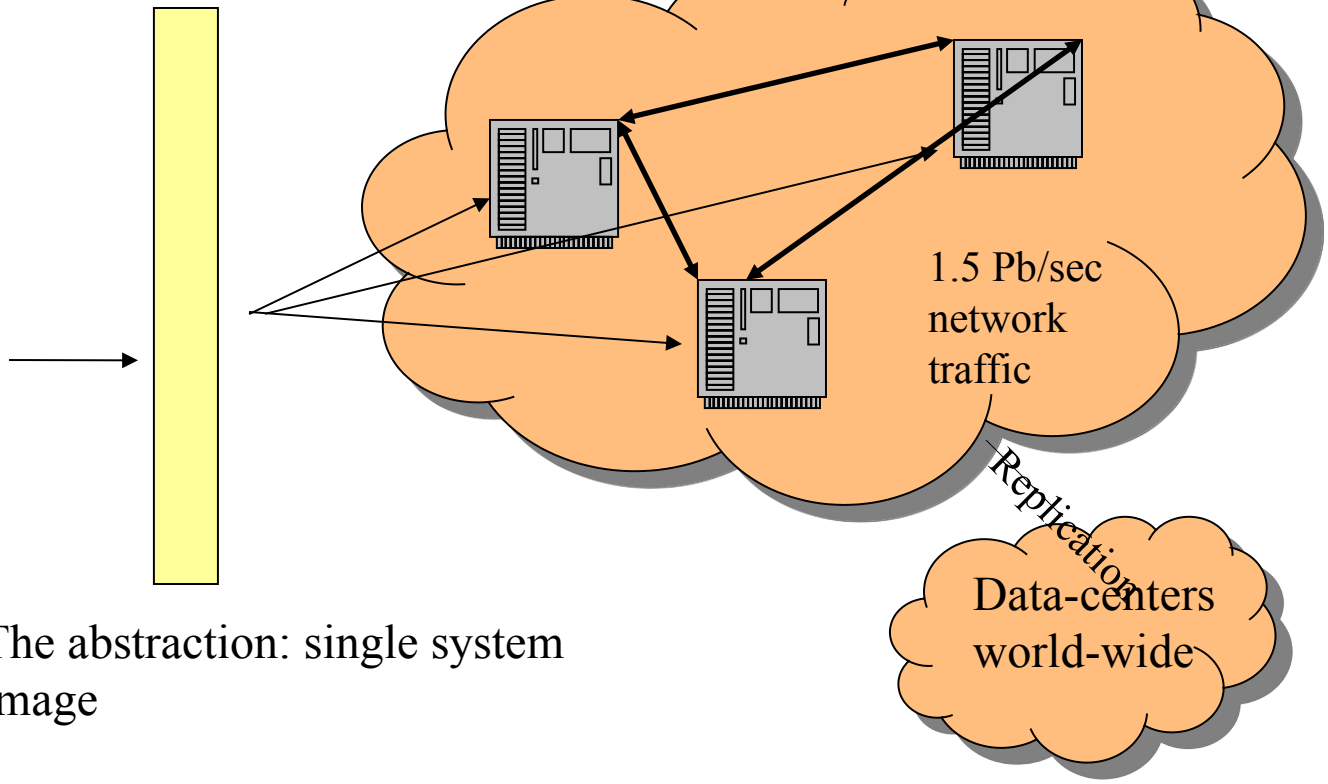
Service Granularity? Application, Component, Object or Request?

Use your “de-hyper” generously!

Warehouse-Scale Computing

Fan-out Architecture

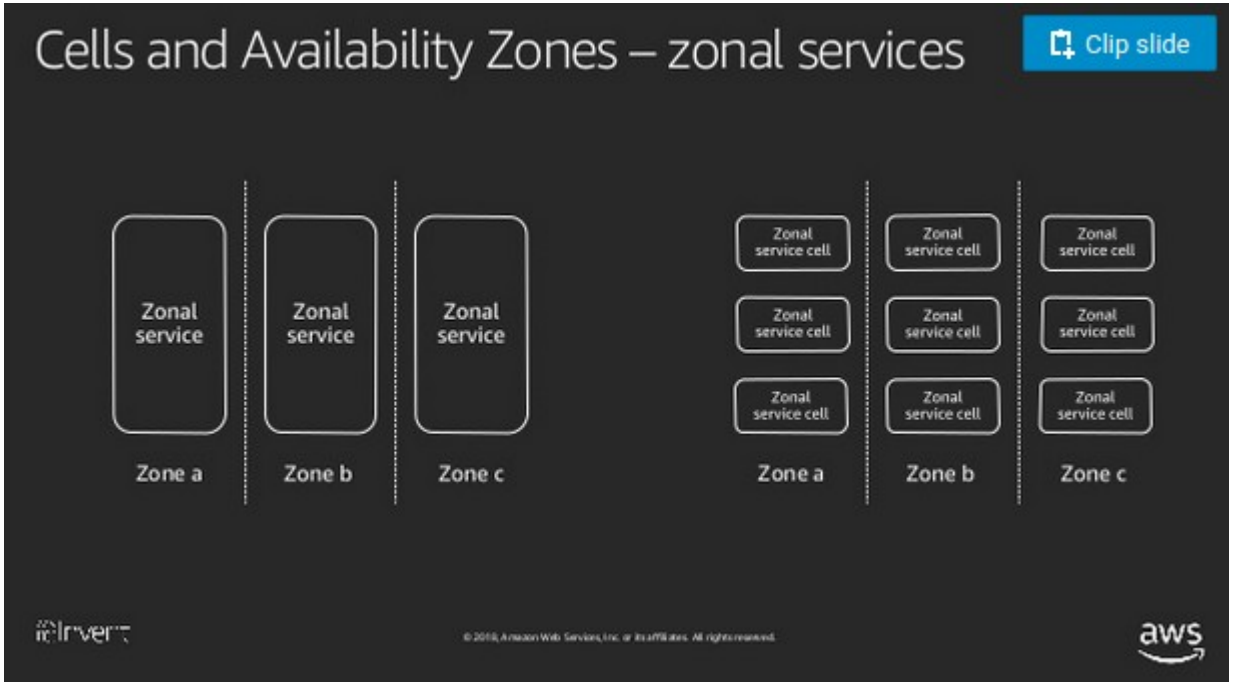
> 100.000 request/sec



The abstraction: single system image

Jeff Dean, " Achieving Rapid Response Times in Large Online Services" http://research.google.com/people/jeff/latency.html and Luiz Barroso, "Warehouse-Scale Computing: Entering the Teenage Decade" http://dl.acm.org/citation.cfm?id=2019527&CFID=39785911&CFTOKEN=33778723

Cloud Computing: Cells and Zones

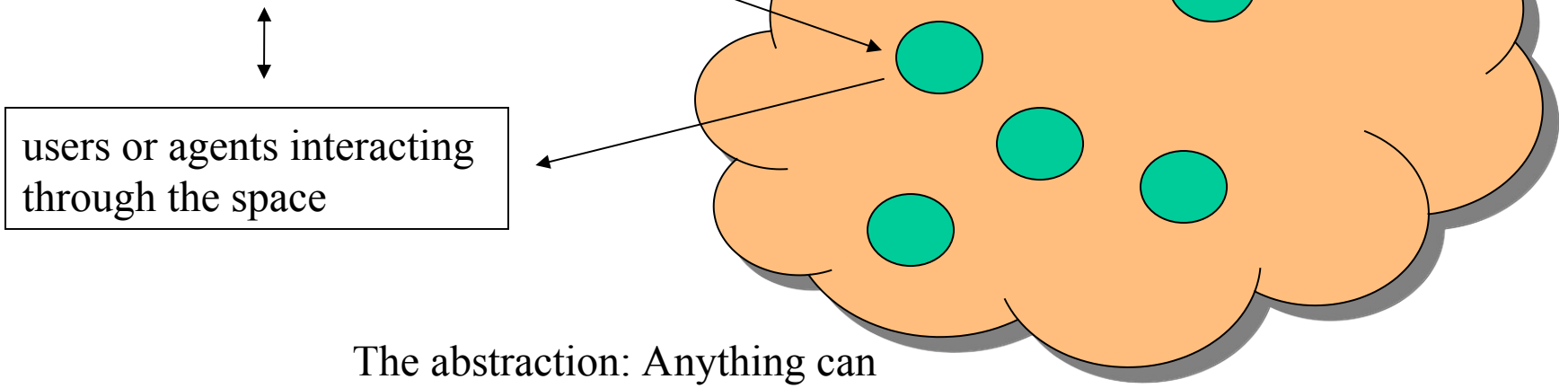


Peter Voss, AWS re:Invent 2018: How AWS Minimizes the Blast Radius of Failures (ARC338) https://www.youtube.com/watch?v=swQbA4zub20&feature=youtu.be

(Tuple) Spaces

A space providing tuple storage

users or agents storing or finding tuples



users or agents interacting through the space

The abstraction: Anything can be stored as long as it is addressable

The worlds largest space is the WWW. Other spaces are WIKI-WIKI collaboration systems or more traditional tuple spaces like tspaces or jspaces. The principle is always the same: a few simple methods (put/take/find) which lets users or machines store or find content. The content itself is returned as a representation of a resource. That‘s why some people call those systems REST (Representational State Transfer Architecture), after a theses from Roy Fielding, the father of http.

How to BUILD middleware?

- Delivery guarantees
- Process models
- Failure models
- Logical time protocols
- Consistency models
- Atomic broadcast
- Fan out architectures
- Partitioning and scaling
- Partition functions
- High availability patterns
- Concurreny patterns
- Replication
- Transactions
- Log-structured merge trees
- Causality and vector clocks
- Consistent hashing
- Distributed security
- Scalability and reliability
- Monitoring, tracing, logging
- Observability
- Scheduling algorithms
- Distributed deadlock detection
- Thundering herds
- Cell architectures
- Latency and bandwidth
- Locality, sharing, connection pooling
- And much more….

Course - Timeline

1. Introduction to DS
2. Theoretical models of distributed systems (queuing theory, process and I/O models)
3. Message protocols (delivery guarantees, causality and reliable broadcast, socket API)
4. Remote procedure calls (classic functions, marshaling, thrift, gRPC, http2.0)
5. Remote objects and frameworks (RMI, EJB)
6. Theoretical foundations of DS (FLP, time, causality and consensus, eventual consistency and optimistic replication)
7. Distributed Services and Algorithms I (balancing, message queues, caching, consistent hashing)
8. Distributed Services II (persistence, transactions, eventual consistency, coordination)
9. Distributed Security (AAA, secure delegation, backend security)
10. Design of Distributed Systems: Methodology and Examples, fan-out architectures
11. System Management in DS (monitoring, chaos monkeys, patterns of resilience)
12. Service Architectures: SOA and Microservices
13. Peer-to-Peer Systems and the Distributed Web (Distributed hashtables, blockchain, onion routing, distributed consensus)
14. Ultra-large-scale Systems (Scalability, performance, network and datacenter design)

Homework for next Session!

READ:

Distributed Systems for fun and profit: (exam-relevant!!)
http://book.mixu.net/distsys/single-page.html

Literature

- Distributed Systems for fun and profit: (exam-relevant!!) http://book.mixu.net/distsys/single-page.html
- Jeff Hodges, Notes on Distributed Systems for Young Bloods
- Brendan Burns, Designing Distributed Systems Patterns and Paradigms for Scalable, Reliable Services
- www.infoq.com on DS frameworks, QCON Videos etc.
- Werner Vogels (Amazon CTO): www.allthingsdistributed.com
- The Source: www.highscalability.com (Todd Hoff)
- Queue.acm.org, free magazine on scalability etc.
- What we do when we talk about distributed systems, Alvaro Videla, http://videlalvaro.github.io/2015/12/learning-about-distributed-systems.html
- Martin Kleppman, Data-Intensive Applications
- Google: Site Reliability Engineering

Resources (Papers)

- Jim Waldo, A note on Distributed Computing (on the transparency dogma in DS)
- Leslie Lamport, Paxos made simple, http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf
- Leslie Lamport, Time, Clocks and the ordering of events in distributed systems, http://research.microsoft.com/en-us/um/people/lamport/pubs/time-clocks.pdf
- Shavit et.al., Atomic Snapshots of Shared Memory, http://people.csail.mit.edu/shanir/publications/AADGMS.pdf
- L.A.Barroso, J Clidaras, U.Hölzle, The Datacenter as a Computer – An Introduction to the Design of Warehouse-Scale Machines, 2nd Edition 2013 (a google book) http://www.morganclaypool.com/doi/pdf/10.2200/S00516ED2V01Y201306CAC024
- Harvest, Yield and Scalable Distributed Systems, Amando Fox, Eric Brewer, (CAP etc.)

Resources (Programming)

- Wolfgang Emmerich, Engineering Distributed Objects (www.distributed-objects.com) With slides and tests.
- Ted Neward, Java Server Side Programming (sockets, servlets etc.) www.manning.com/neward
- www.swarm.org, portal for swarm programming. Used also as simulation tools for research in economics and finance
- Apache Thrift, http://jnb.ociweb.com/jnb/jnbJun2009.html
- Bjorn Hansen, Real World Web Performance & Scalability

Resources (Basics)

- Coulouris, e.al., Distributed Systems
- Andrew Tanenbaum, Maarten van Steen, Distributed Systems. Get this one or Coulouris for a long term effect .
- Ken Birman, Building secure and reliable Network Applications
- Grey/Reuter, Transaction Processing
- Jiro/Federated Management Architecture (FMA)
- M.Cavage, There's Just No Getting around It: You're Building a Distributed System, ACM Queue, April. 2013, http://portal.acm.org/ft_gateway.cfm?id=2482856&type=pdf

Resources (Theory)

- Designing Distributed Systems, A Conversation with Ken Arnold, Part III, http://www.artima.com/intv/distribP.html , shows importance of failures and state in DS
- The Paradigm Shift from Algorithms to Interaction, Peter Wegner, 1996, a provocative short essay on why interactive systems are much more powerful than turing machines. Shows that DS is more than just concurrency and remoteness. The basics of emergence and non-algorithmic behavior. Good for agent systems as well.
- Phillip J. Windley, Digital Identity,
  Contains architecture of identity repositories including federation aspects. Network effects and its effects against bilateral identity management.
- Nancy A. Lynch, Distributed Algorithms (proofs and concepts)
- Stability and topology of scale-free networks under attack and defense strategies, Lazaros K. Gallos u.a. http://xxx.lanl.gov/pdf/cond-mat/0505201
- Albert-Laszlo Barabasi, Linked. Investigates small worlds, scale free networks etc. Basically moves from random networks to hub/spoke architectures. Discovered how the WWW space is organized (in/out/core/islands etc.). A must read for everybody interested in the effects of topology (e.g. on virus spreads)

Resources High Scalability

Reading list of HS papers,
https://github.com/binhnguyennus/awesome-scalability

Resources (Web)

- Tim Oreilly's famous article on Web2.0: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
- **Gartner's 2006 Emerging Technologies Hype Cycle Highlights Key Technology Themes** http://www.gartner.com/it/page.jsp?id=495475

Resources (Events, Simulation)

- Simjava, discrete event simulation package. Tutorial at: http://www.dcs.ed.ac.uk/home/simjava/tutorial/

- **GridSim,** Grid Simulation Package, http://www.gridbus.org/gridsim/gridsim2.2/