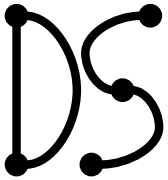


Research Group
Distributed Systems

Distributed Systems

MapReduce & Dynamo

Olaf Landsiedel

Last Time

- TOR
- Google File System

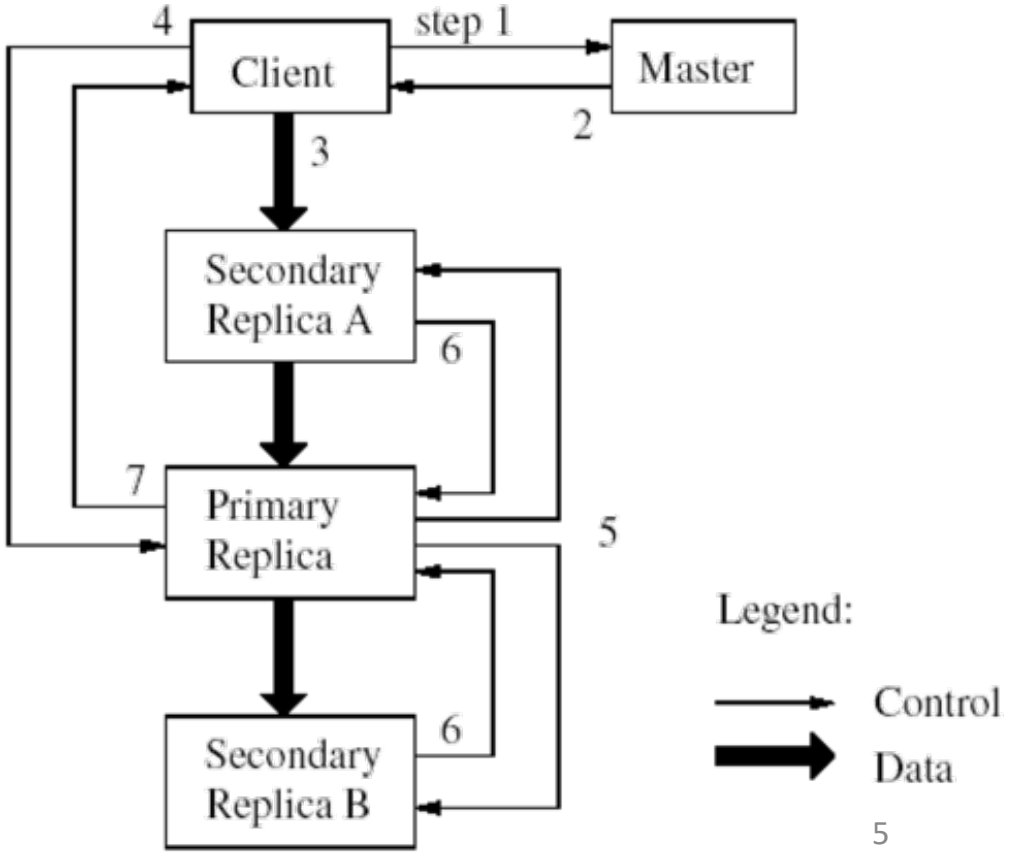
Today

- Complete GFS
- Two more “real” systems
  - Google’s MapReduce
    - Distributed processing of large data sets
  - Amazon Dynamo
    - Large-scale data storage

GFS

Mutations

- Mutation = write or append
  - must be done for all replicas
- Goal: minimize master involvement
- Lease mechanism:
  - master picks one replica as primary; gives it a “lease” for mutations
  - primary defines a serial order of mutations
  - all replicas follow this order
- Data flow decoupled from control flow



Relaxed Consistency Model

- “Consistent” = all replicas have the same value
- “Defined” = replica reflects the mutation, consistent

- Some properties:
  - concurrent writes leave region consistent, but possibly undefined
    - -> unsure which write was applied first and which second
  - failed writes leave a region inconsistent
- Some work has moved into the applications:
  - e.g., self-validating, self-identifying records

Master's responsibilities (1/2)

- Metadata storage
- Namespace management/locking
- Periodic communication with chunk-servers
  - give instructions, collect state, track cluster health
- Chunk creation, re-replication, rebalancing
  - balance space utilization and access speed
  - spread replicas across racks to reduce correlated failures
  - re-replicate data if redundancy falls below threshold
  - rebalance data to smooth out storage and request load

Master's responsibilities (2/2)

- Garbage Collection
  - simpler, more reliable than traditional file delete
  - master logs the deletion, renames the file to a hidden name
  - lazily garbage collects hidden files
- Stale replica deletion
  - detect "stale" replicas using chunk version numbers

Fault Tolerance

- High availability
  - fast recovery
    - master and chunkservers restartable in a few seconds
  - chunk replication
    - default: 3 replicas.
  - shadow masters
- Data integrity
  - checksum every 64KB block in each chunk
    - In addition to checksums / FEC of the underlying file system
      - Which can have even more frequent checksums

Conclusion

- GFS demonstrates how to support large-scale processing workloads on commodity hardware
  - design to tolerate frequent component failures
  - optimize for huge files that are mostly appended and read
  - feel free to relax and extend FS interface as required
  - go for simple solutions (e.g., single master)

- GFS has met Google's storage needs…
  - it must be good!

GFS Discussion

- Summarize GFS

- olafland.polldaddy.com/s/gfs
  - GFS is highly fault tolerant
  - GFS has a single point of failure
  - Writes in GFS are send to the master node
  - GFS allows any degree of redundancy

olafland.polldaddy.com/s/gfs

GFS Discussion

- olafland.polldaddy.com/s/gfs
  - GFS is highly fault tolerant
    - yes
  - GFS has a single point of failure
    - No: shadow masters
  - Writes in GFS are send to the master node
    - No: directly to the replica
  - GFS allows any degree of redundancy
    - yes

Map Reduce

Why distributed computations?

- How long to do word counting of 1 TB on one computer?
  - One computer can read ~60MB/s from disk
  - Takes ~1 days!!
- Google indexes 100 billion+ web pages
  - 100 * 10^9 pages * 20KB/page = 2 PB
- Large Hadron Collider is expected to produce 15 PB every year!

Solution: use many nodes!

- 1000 nodes potentially give 1000X speedup

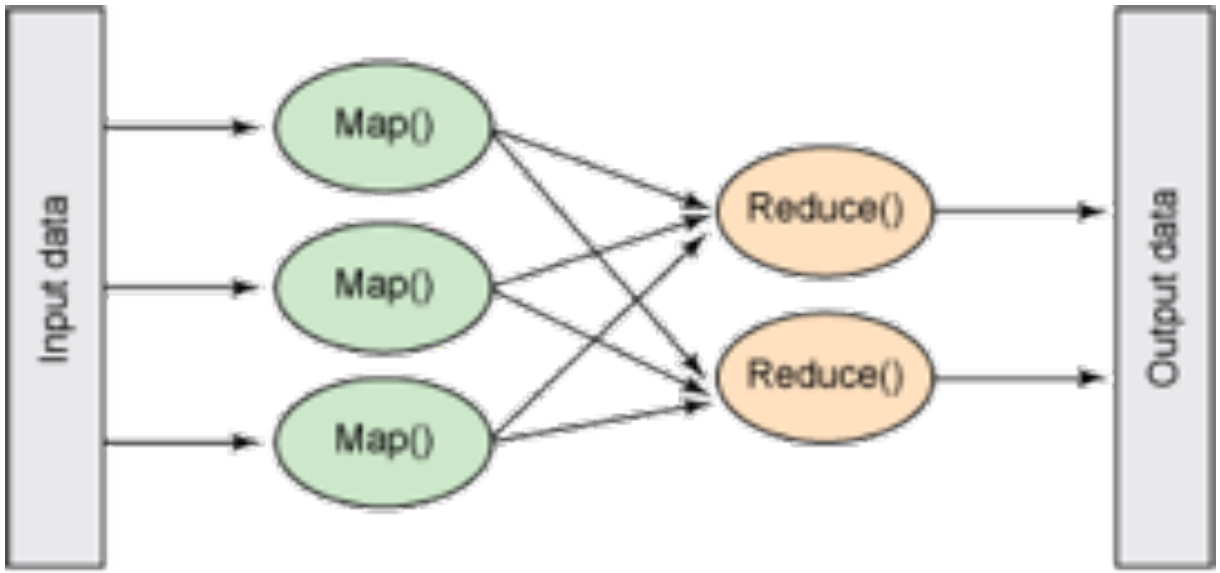
MapReduce

- A programming model for large-scale computations
  - Process large amounts of input, produce output
  - No side-effects or persistent state (unlike file system)

- MapReduce is implemented as a runtime library:
  - automatic parallelization
  - load balancing
  - locality optimization
  - handling of machine failures

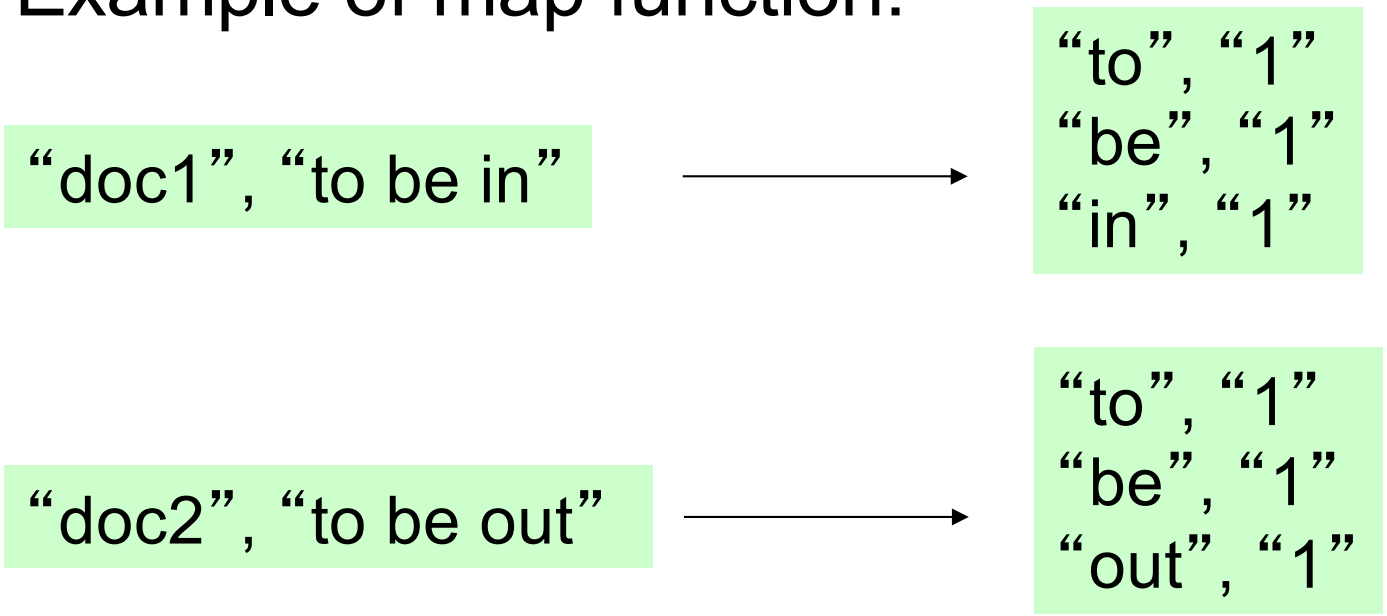
MapReduce design

- Input data is partitioned into M splits
- Map: extract information on each split
  - Each Map produces R partitions
- Shuffle and sort
  - Bring M partitions to the same reducer
- Reduce: aggregate, summarize, filter or transform



Example: Word Count in Web Pages

- Main step 1: Map
  - Map parses documents into words
    - key = document URL
    - value = document contents
  - Example of map function:



Example: word frequencies

"to", "1"
"be", "1"
"in", "1"

"to", "1"
"be", "1"
"out", "1"

- Main Step 2
  - Reduce: computes sum for a word across file

key = "be"
values = "1", "1"
→ "2"

key = "to"
values = "1", "1"
→ "2"

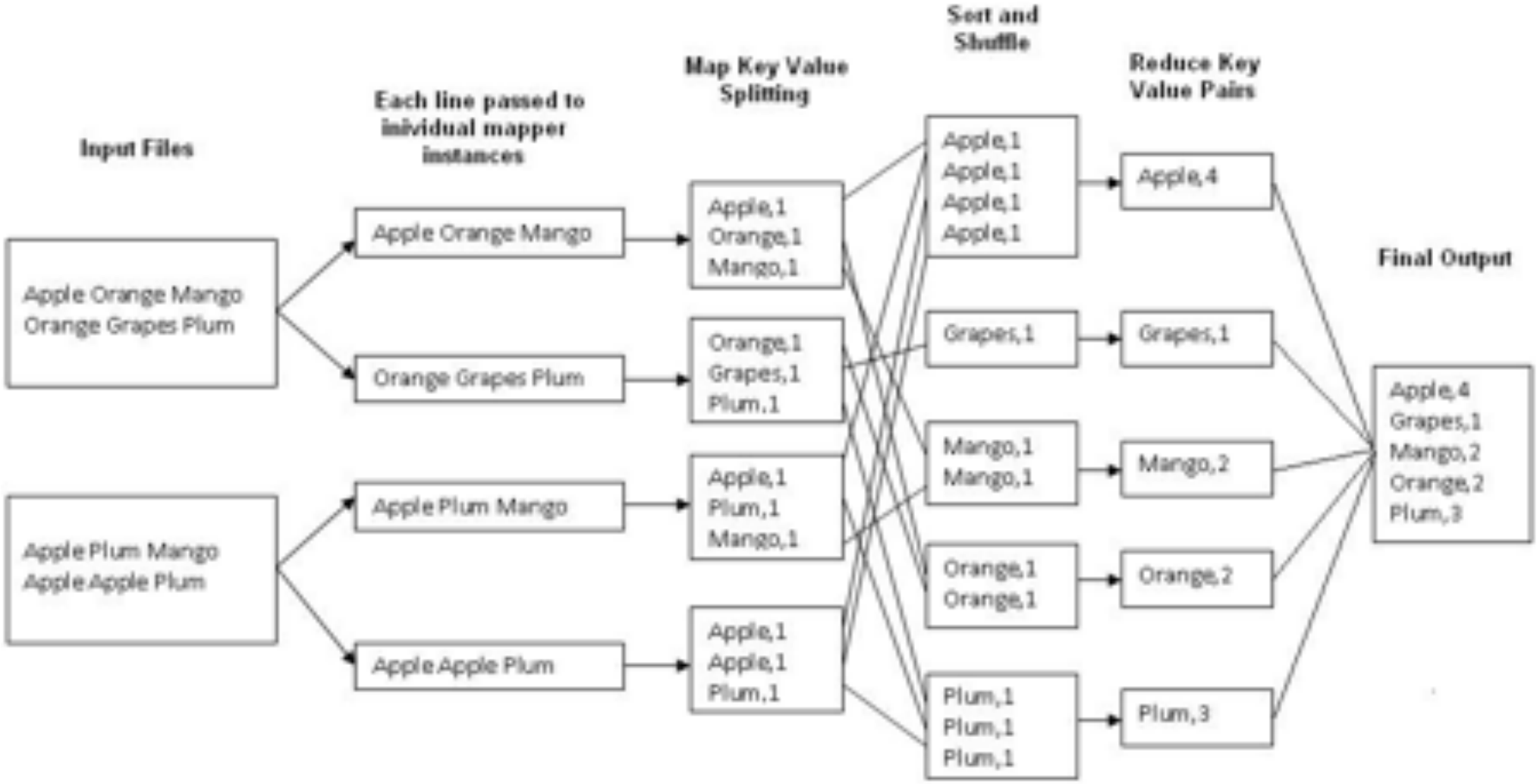
key = "out"
values = "1"
→ "1"

key = "in"
values = "1"
→ "1"

- Output of reduce saved

"be", "2"
"to", "2"
"in", "1"
"out", "1"

One more example



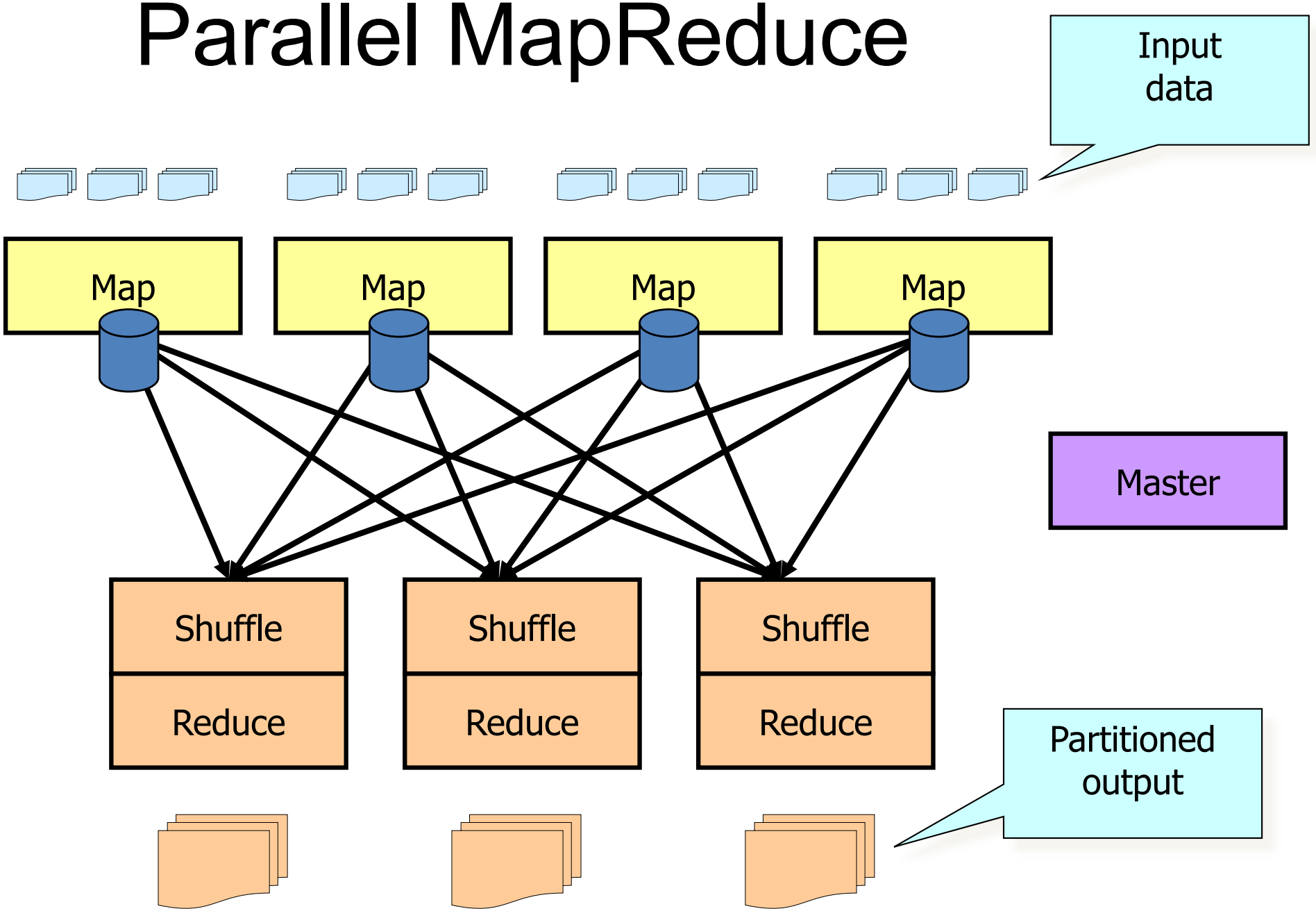
MapReduce is widely applicable

- Distributed grep
- Document clustering
- Web link graph reversal
- Detecting approx. duplicate web pages
- …

MapReduce scheduling

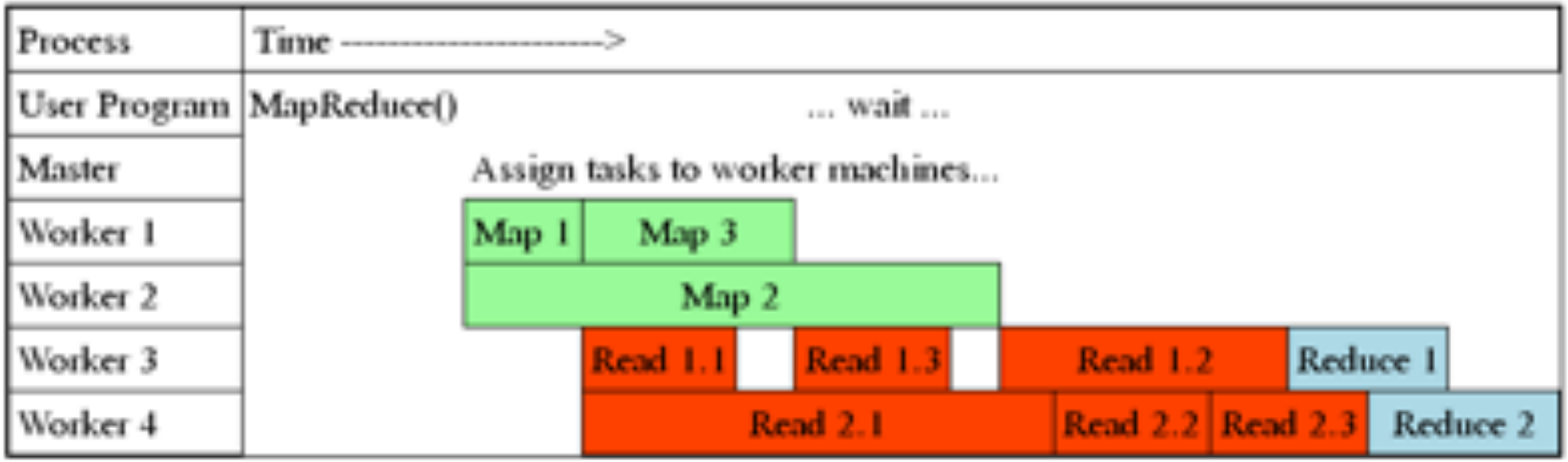
- One master, many workers
  - Input data split into *M* map tasks (e.g. 64 MB)
  - *R* reduce tasks
  - Tasks are assigned to workers dynamically
  - Often: *M*=200,000; *R*=4,000; workers=2,000

Parallel MapReduce



Load Balance and Pipelining

- Fine granularity tasks: many more map tasks than machines
  - Minimizes time for fault recovery
  - Can pipeline with map execution



Fault tolerance via re-execution

On worker failure:

- Re-execute completed and in-progress map tasks
- Re-execute in progress reduce tasks
- Task completion committed through master

On master failure:

- State is checkpointed to GFS: new master recovers & continues

Avoid straggler using backup tasks

- Slow workers significantly lengthen completion time
  - Other jobs consuming resources on machine
  - Bad disks with soft errors transfer data very slowly
  - Weird things: processor caches disabled (!!)
  - An unusually large reduce partition?

- Solution: Near end of phase, spawn backup copies of tasks
  - Whichever one finishes first "wins”
  - Compare to “end game” in BitTorrent

- Effect: Dramatically shortens job completion time

Programming Framework

- MapReduce is a programming framework
  - Some task are provided by the user/programmers
  - Others by the framework
- MapReduce has four main phases:
  - 1. Map: Mappers provided by the user/programmers
  - 2. Shuffle and 3. sort: provided by the framework
  - 4. Reduce: Reducer must be supplied by user/programmers
  - Overall framework: provided: load balancing, pipeling, restart of failed mappers, reducers, etc.

Map Reduce

- Goal of Map Reduce
  - Fast distributed data processing

- Originally developed by Google
  - Similar things from Facebook, Yahoo, …
  - Example: Hadoop
    - Open source implementation for MapReduce
    - Used by Facebook, Yahoo, and many others

Map Reduce

- Summarize Map Reduce

- olafland.polldaddy.com/s/mapreduce
  - Map reduce is a good application for
    - Counting words in files
    - Unique items counting
    - Managing bank accounts
    - Online shopping carts

olafland.polldaddy.com/s/mapreduce

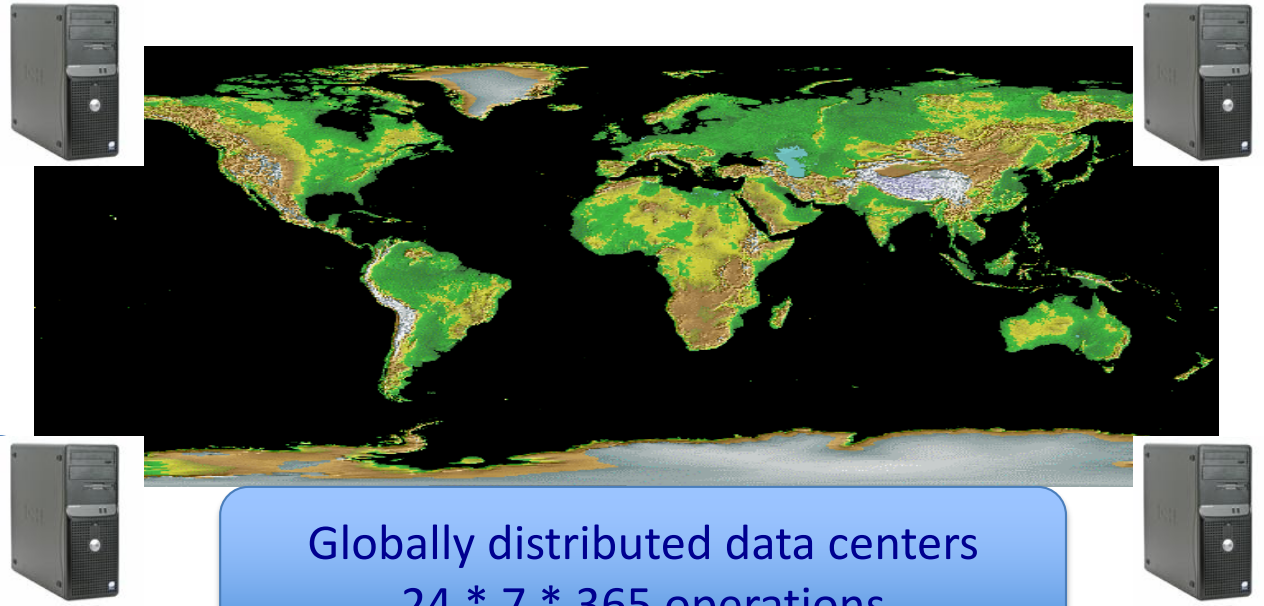
Map Reduce

- olafland.polldaddy.com/s/mapreduce
  - Map reduce is a good application for
    - Counting words in files: yes
    - Unique items counting: yes
    - Managing bank accounts: no
    - Online shopping carts: no

Amazon Dynamo

Amazon Online Shopping

Tens of millions of customers



Tens of thousands of servers

Globally distributed data centers
24 * 7 * 365 operations

Financial consequences

Customer Trust

DATA MGMT

Performance

Reliability

Efficiency

Scalability

Scale

- Amazon is busy during the holidays
  - Shopping cart: tens of millions of requests for **3 million checkouts in a single day**
  - Session state system: **100,000s of concurrently active sessions**

Failure in the Data Center

- Failure is common
  - Small but significant number of server and network failures at all times
    - See lecture on fault tolerance
- Goal
  - “Customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados.”

General Design Challenges

- Data center services must address
  - Availability
    - Service must be accessible at all times
  - Scalability
    - Service must scale well to handle customer growth & machine growth
  - Failure Tolerance
    - With thousands of machines, failure is the default case
  - Manageability
    - Must not cost a fortune to maintain

Amazon Dynamo

- A highly available key-value storage system
  - put(), get() interface
    - Like a hash table and a DHT (see previous lecture)
  - Sacrifices consistency for availability
    - Provides eventual consistency
      - What was this?
  - Provides storage for some of Amazon's key products
    - e.g., shopping carts, bestseller lists, …
  - Design shows similarities to Chord
    - Adapted to data center and application requirements

Dynamo: The Big Picture

Consistency Model

- Eventual consistency
- “Always writable”
  - Can always write to shopping cart
  - Pushes conflict resolution to reads
- Application-driven conflict resolution
  - e.g., merge conflicting shopping carts
  - Or Dynamo enforces last-writer-wins

Data Versioning

- Updates generate a new timestamp
  - Vector clocks
- Eventual consistency
  - Multiple versions of the same object might co-exist
- Syntactic Reconciliation
  - System might be able to resolve conflicts automatically: using timestamp
- Semantic Reconciliation
  - Conflict resolution pushed to application
    - Recall Bayou?

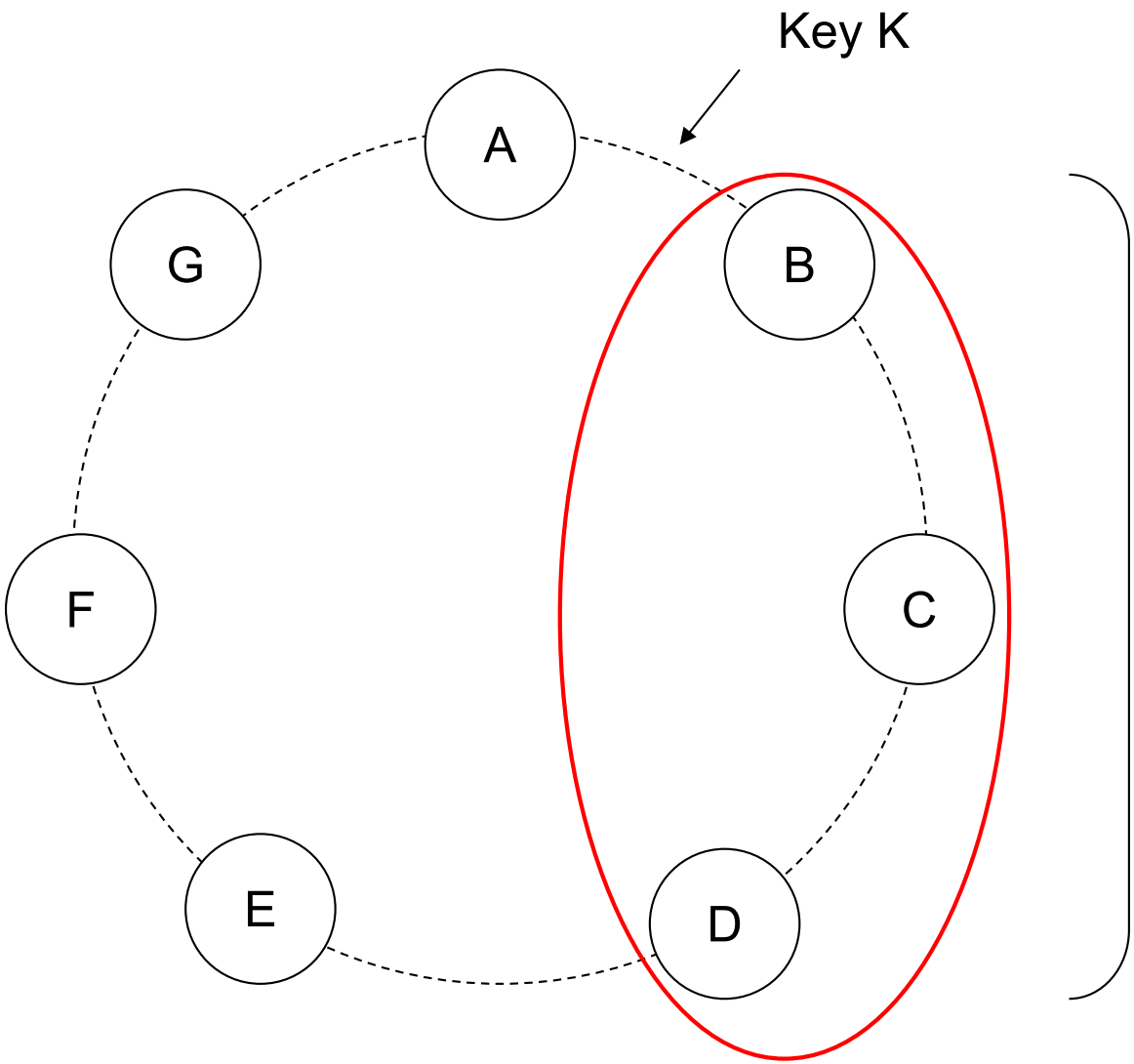
Data Versioning

- put() can return before update is applied to all replicas
- Subsequent get()s can return older versions
  - This is okay for shopping carts
    - Deleted items can resurface
- A vector clock is associated with each object version
  - Comparing vector clocks can determine whether two versions are parallel branches or causally ordered

Data Partitioning & Replication

- Use hashing
- Similar to Chord
  - Each node gets an ID from the space of keys
  - Nodes are arranged in a ring
  - Data stored on the first node clockwise of the current placement of the data key
- Replication
  - Preference lists of N nodes following the associated node

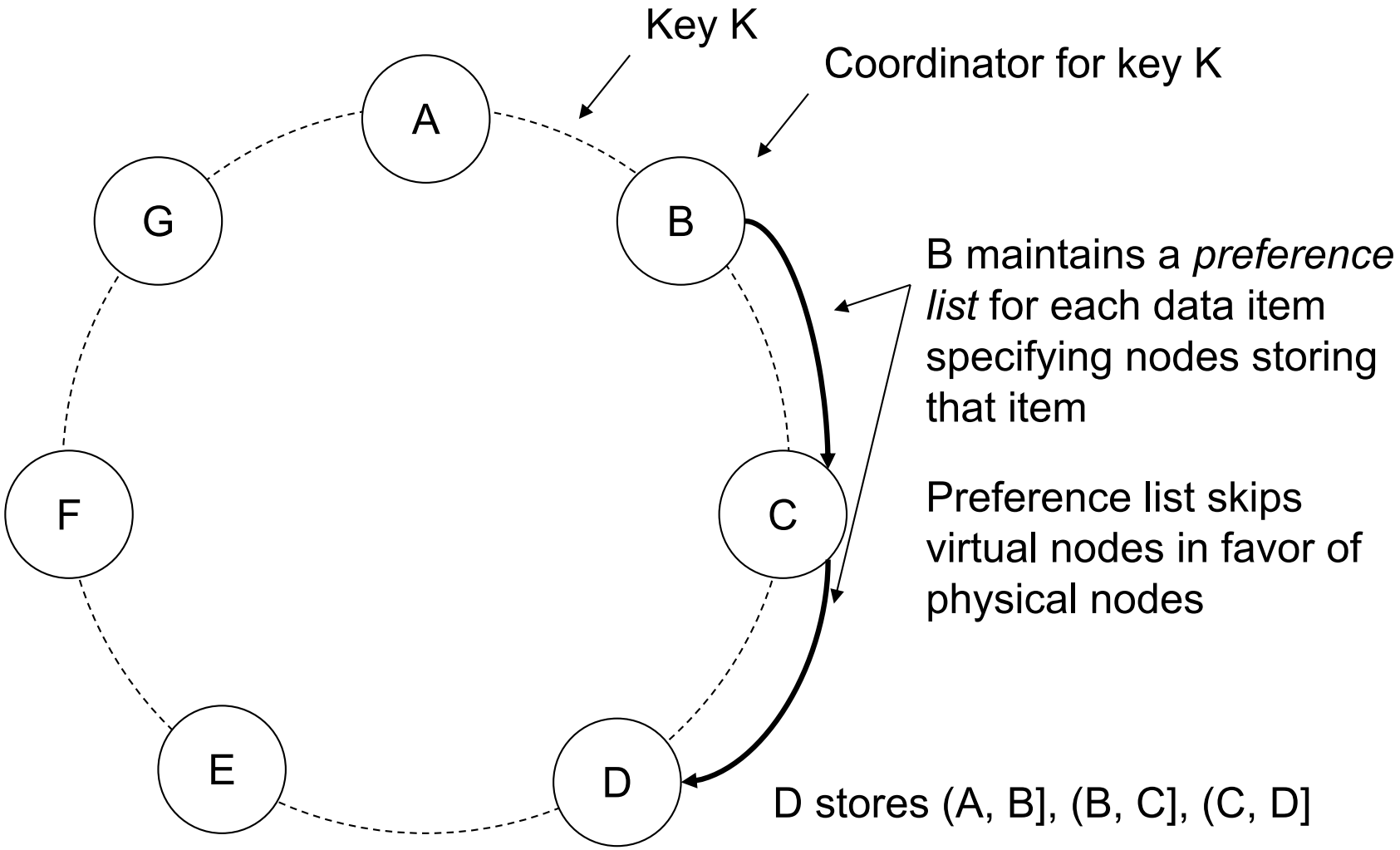
Storage: Variant of Chord DHT



Each node is assigned to multiple points in the ring
e.g., B, C, D store keyrange (A, B)
# of points can be assigned based on node's capacity

If node becomes unavailable, load is distributed to others

Replication



Virtual Nodes on the Chord Ring

- A problem with the Chord scheme
  - Nodes placed randomly on ring
  - Leads to uneven data & load distribution
- In Dynamo
  - Use “virtual nodes”
  - Each physical node has multiple virtual nodes
    - More powerful machines have more virtual nodes
  - Distribute virtual nodes across the ring

Routing

- Chord
  - Routing (also finger) table size: log(n)
    - Scales to millions of nodes
  - Lookup: use finger table for log(n) hops
- Dynamo
  - Routing table size: n
    - Fully meshed
    - Scales to thousands of nodes
    - Perfect for data centers
  - Lookup: one hop
    - Strong performance

“Quorum-likeness”

- get() & put() driven by two parameters:
  - R: the minimum number of replicas to read
  - W: the minimum number of replicas to write
- R + W > N yields a “quorum-like” system
  - See lecture on replication
- Latency is dictated by the slowest R (or W) replicas

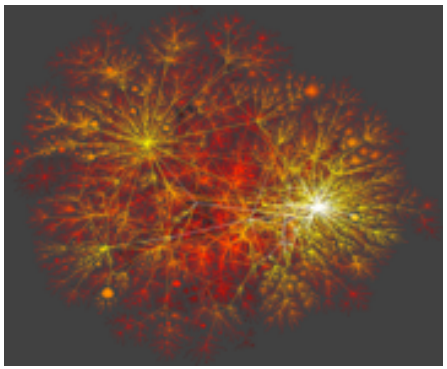
Discussion

- Dynamo ingredients
  - Consistency?
    - Eventual Consistency
  - Event sorting?
    - Vector Clocks
  - Determine valid data when reading from replications?
    - Quorum
  - Data storage?
    - Chord like
- Now you see how the different mechanisms discussed in class can assemble a DS

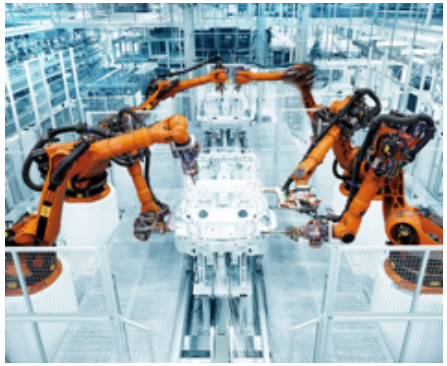
Research Group: Distributed Systems

- Research
  - Distributed Systems
  - Networked Systems
  - Internet of Things (IoT)
  - Wireless Sensor Networks
  - Network Security
  - Applied AI in IoT
  - Blockchain

Cloud Computing



Internet



Mission-Critical Internet of Things

Cooperative Systems

Teaching Bachelor (in German)

1. Operating- and Communication Systems (2. Semester)
   – Betriebs- und Kommunikationssysteme
2. Lab IT-Security (each term)
3. Bachelor Seminar
4. Bachelor thesis

   in the field of distributed systems

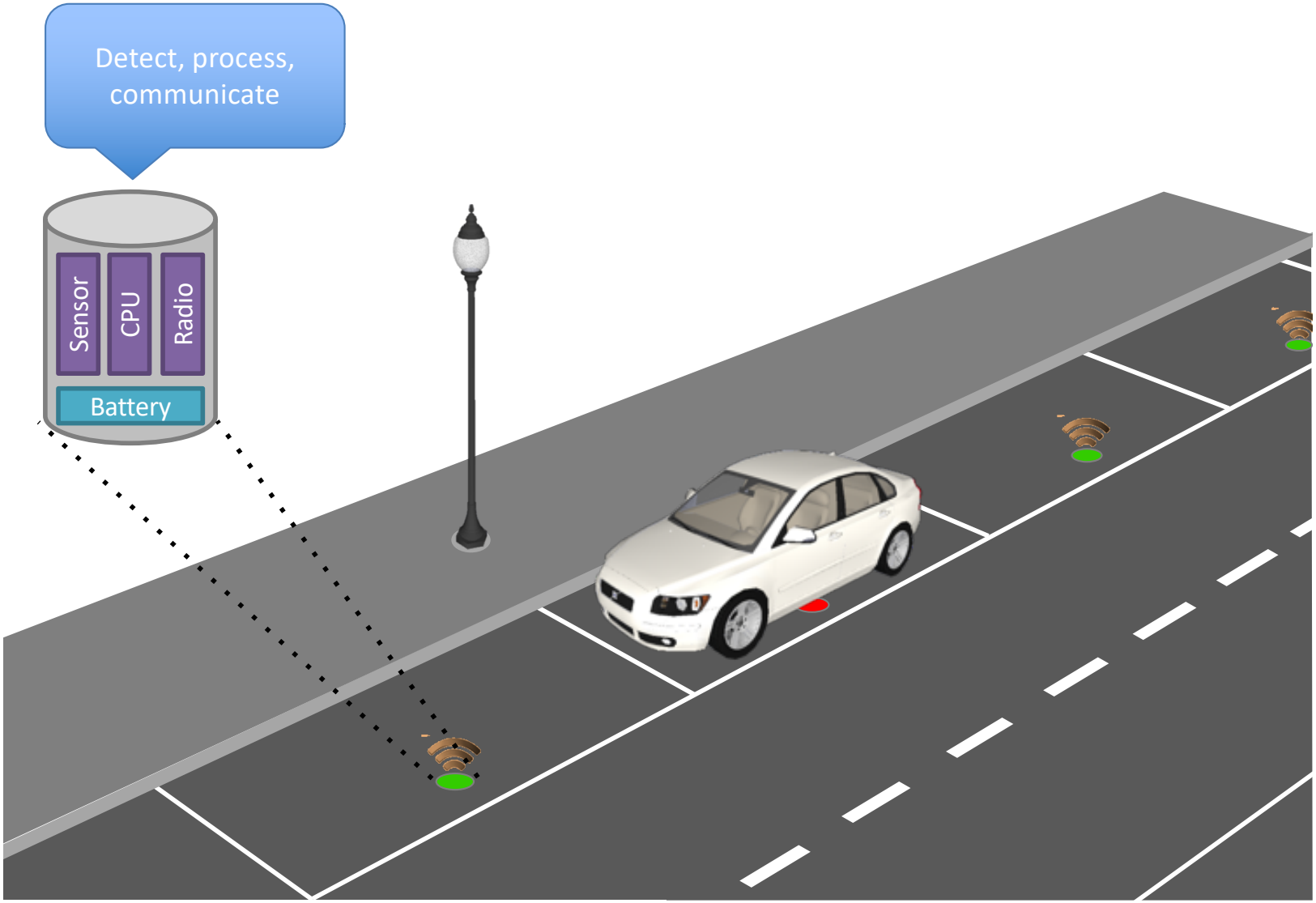
Teaching Master (in English)

1. Distributed Systems (8 ECTS), winter term
   - How to build large-scale systems (Internet, cloud computing, …)
   - Make them scale, fault-tolerant, adaptive, …
2. Mobile Internet & Internet of Things (8 ECTS), summer term
   - Mobile Internet and Wireless Networks
   - Internet of Things: Protocols, Applications
3. Advanced Computer Networks and Network Security (8 ECTS)
   - Network Security
   - Dive deep into the network stack
4. Projects, Seminars, Thesis
   - Master projects and seminars in the field
   - Master thesis…

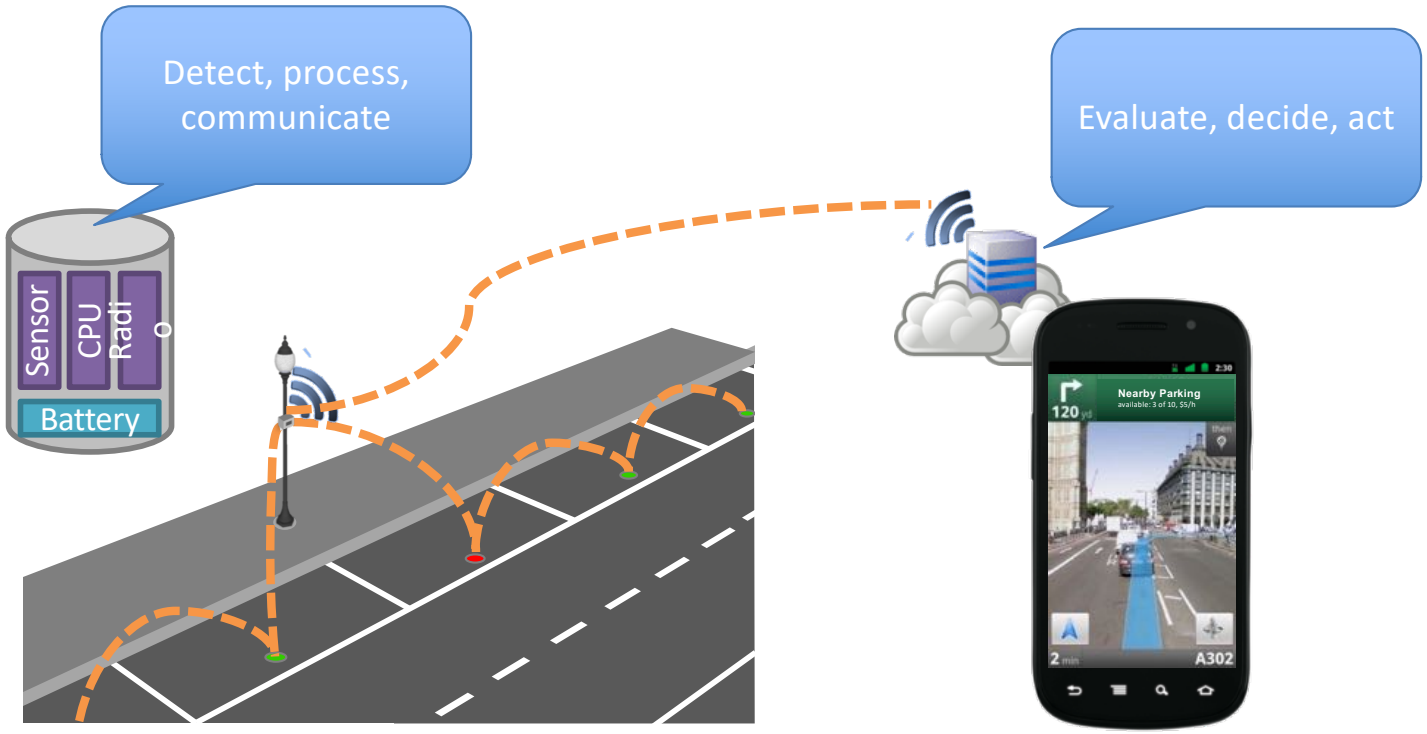
Practical courses with strong systems focus

Where is the next free parking spot?


2:30
Nearby Parking
120 yd
available: 3 of 10, $5/h
then
2 min
A302


Detect, process, communicate
Sensor
CPU
Radio
Battery

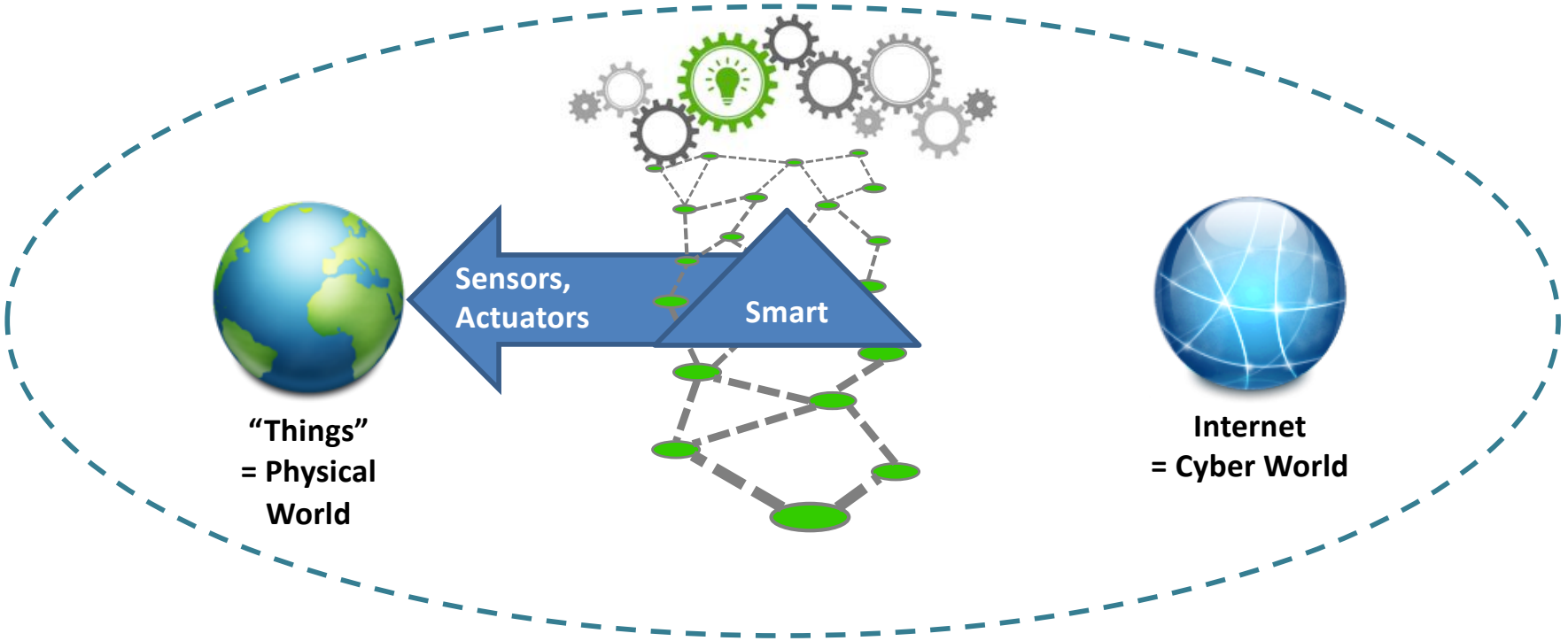
Vision



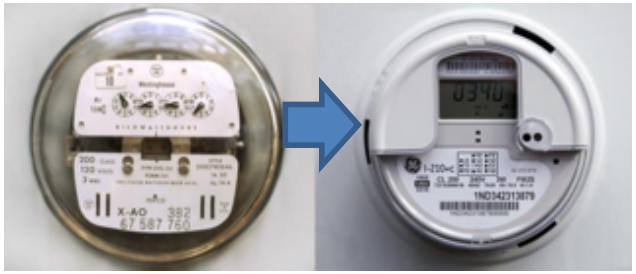
Internet of Things (IoT):
Network of interacting sensors and actuators
Things that Communicate and Make Decisions

30% o k.org)

Internet of Things

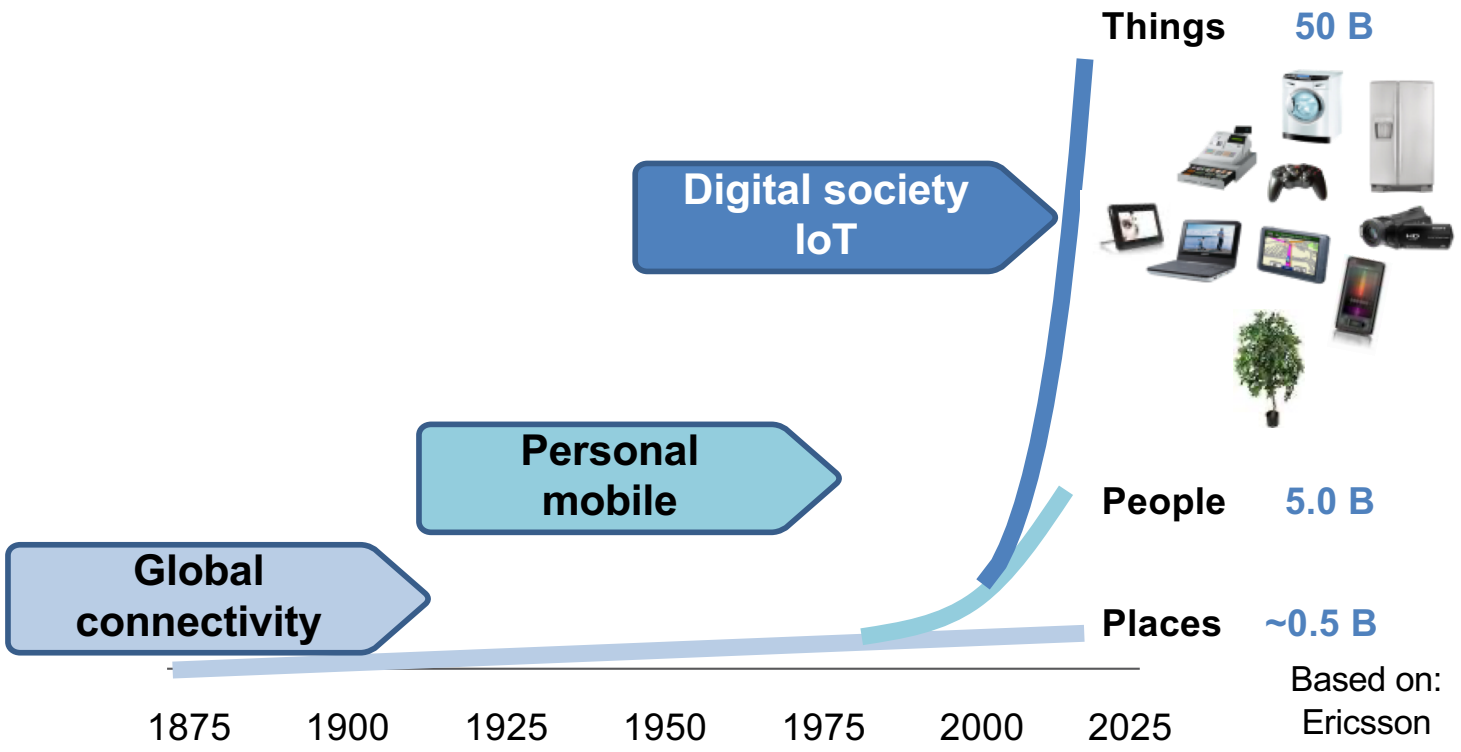


It is happening already

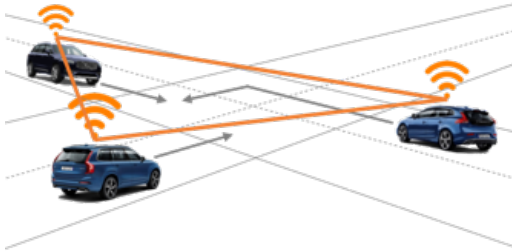


Dumb devices get connected & smart
They become IoT devices

Tomorrow? 50 Billion Connected Devices



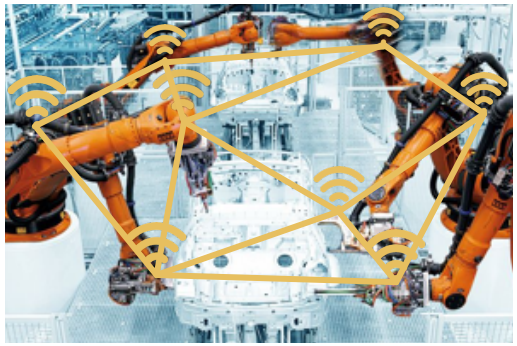
Imagine the future Internet of Things (IoT) …



Cooperative Driving

Vehicles coordinate to

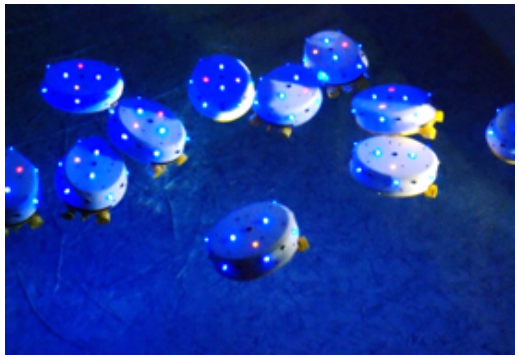
- drive safer,
- drive sustainable



Wireless Factories

Thousands of Wireless Sensor and Actuators

- Flexible factories (Industry 4.0)
- Reduced costs, no-wear/tear

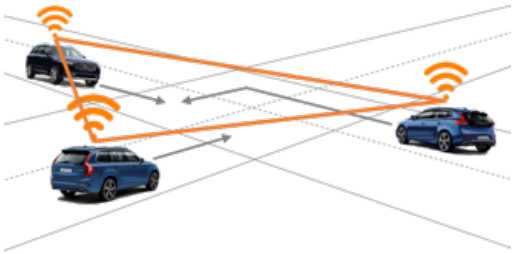


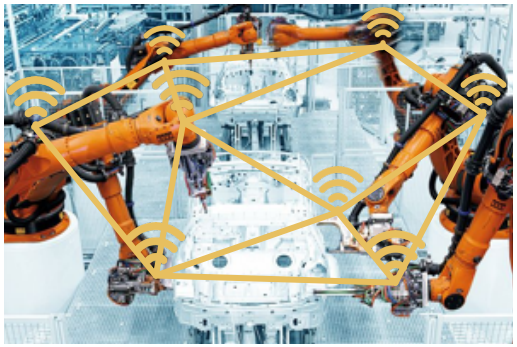
Smart Robots

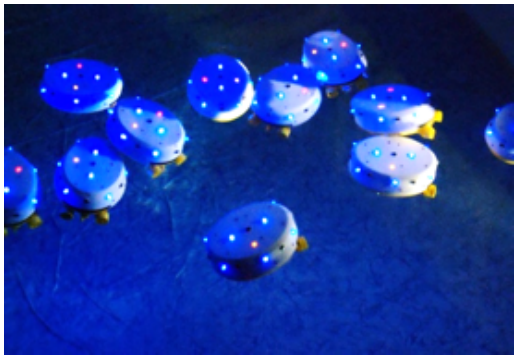
Communicating to

- Support humans
- Act in hazardous environments

Internet of Things

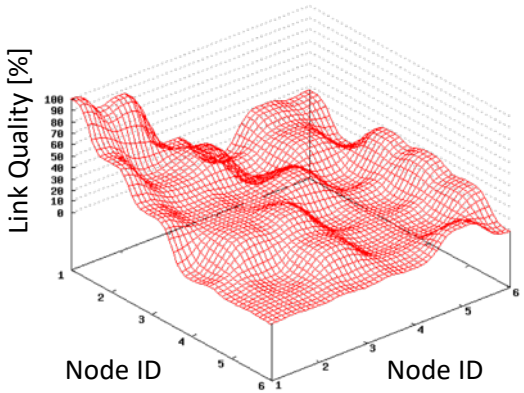






- Networks: Low-Latency Wireless including 5G
- Data: “Data is the new oil” -> Artificial Intelligence
- Applications: Smart City, Digital Society, Cooperative Driving, Industry 4.0, Marine Systems, …

Challenges



Networking

- Dynamic wireless channel
- Interference
- Moving objects

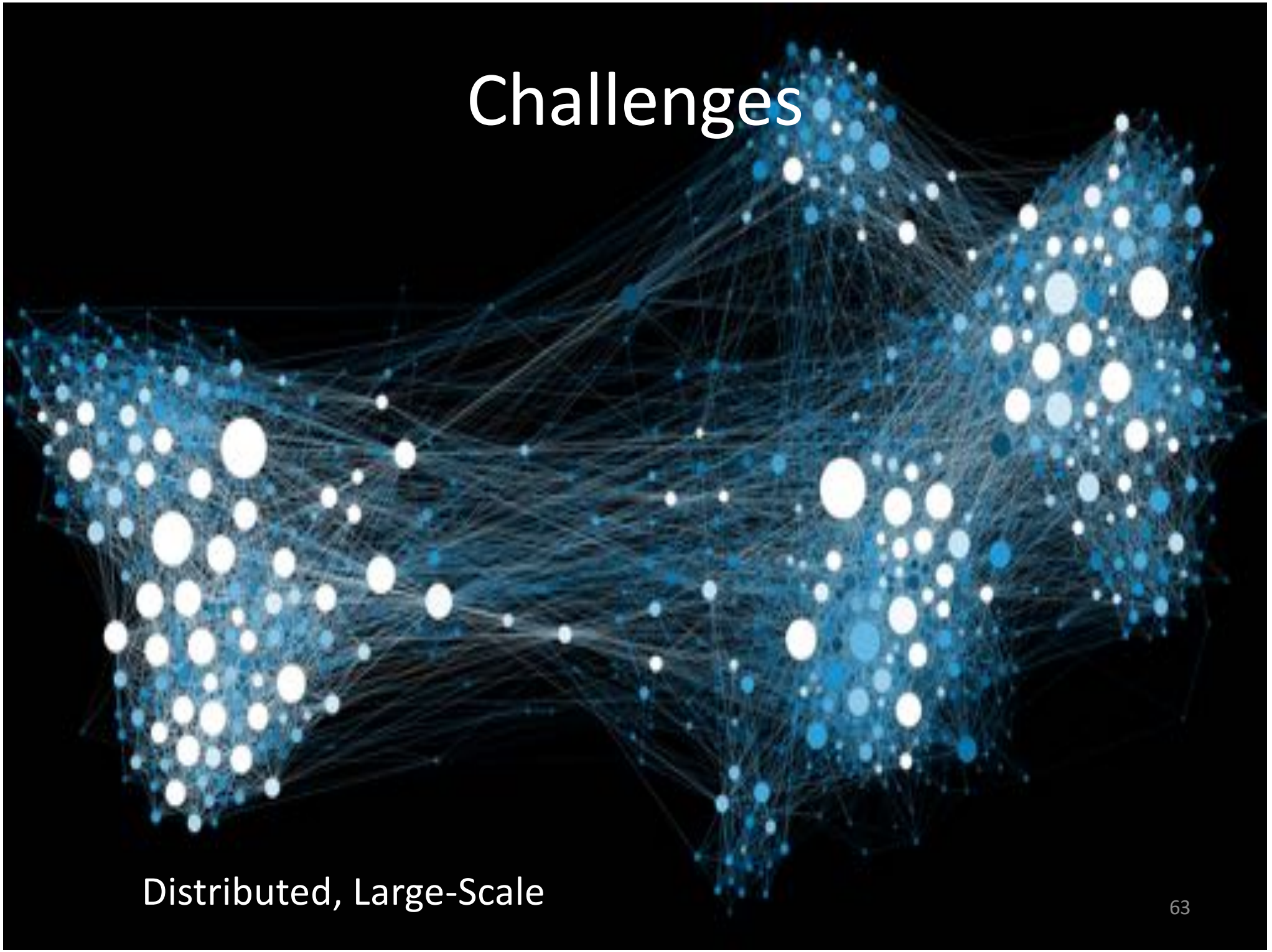
Resource Constraints

- Compute Power
- Energy

Smart

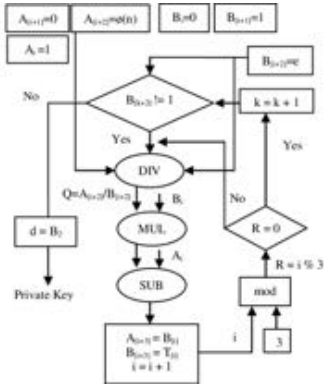
- Distributed Data Analytics
- AI in IoT
- Agreement

Challenges

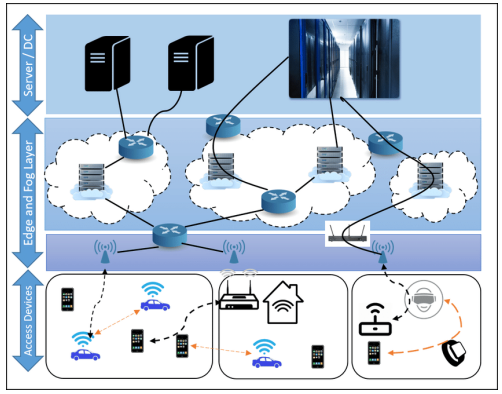


Distributed, Large-Scale

Data-Driven, Experimental Systems Research



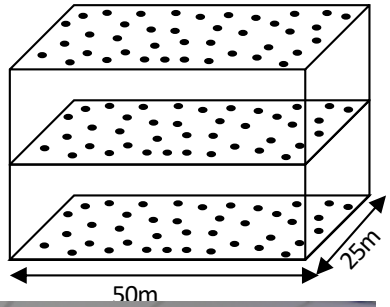
Practical Algorithms



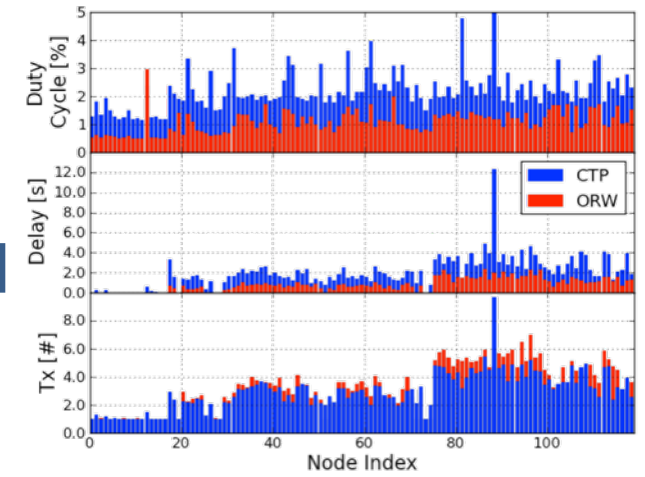
System Architectures

Implementation

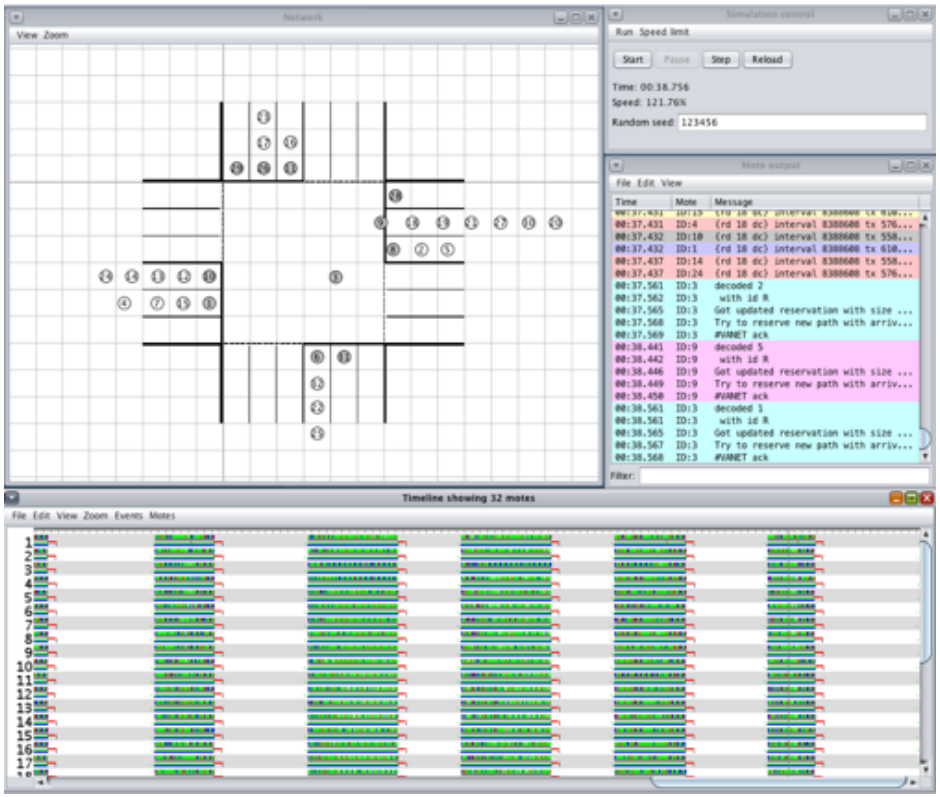
Data-Driven, Experimental Systems Research

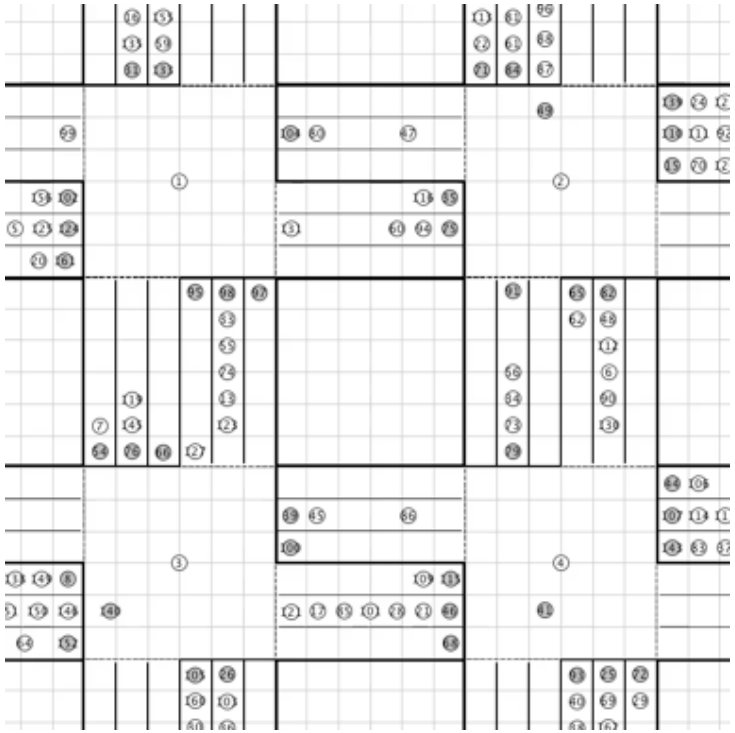


GBs of Traces

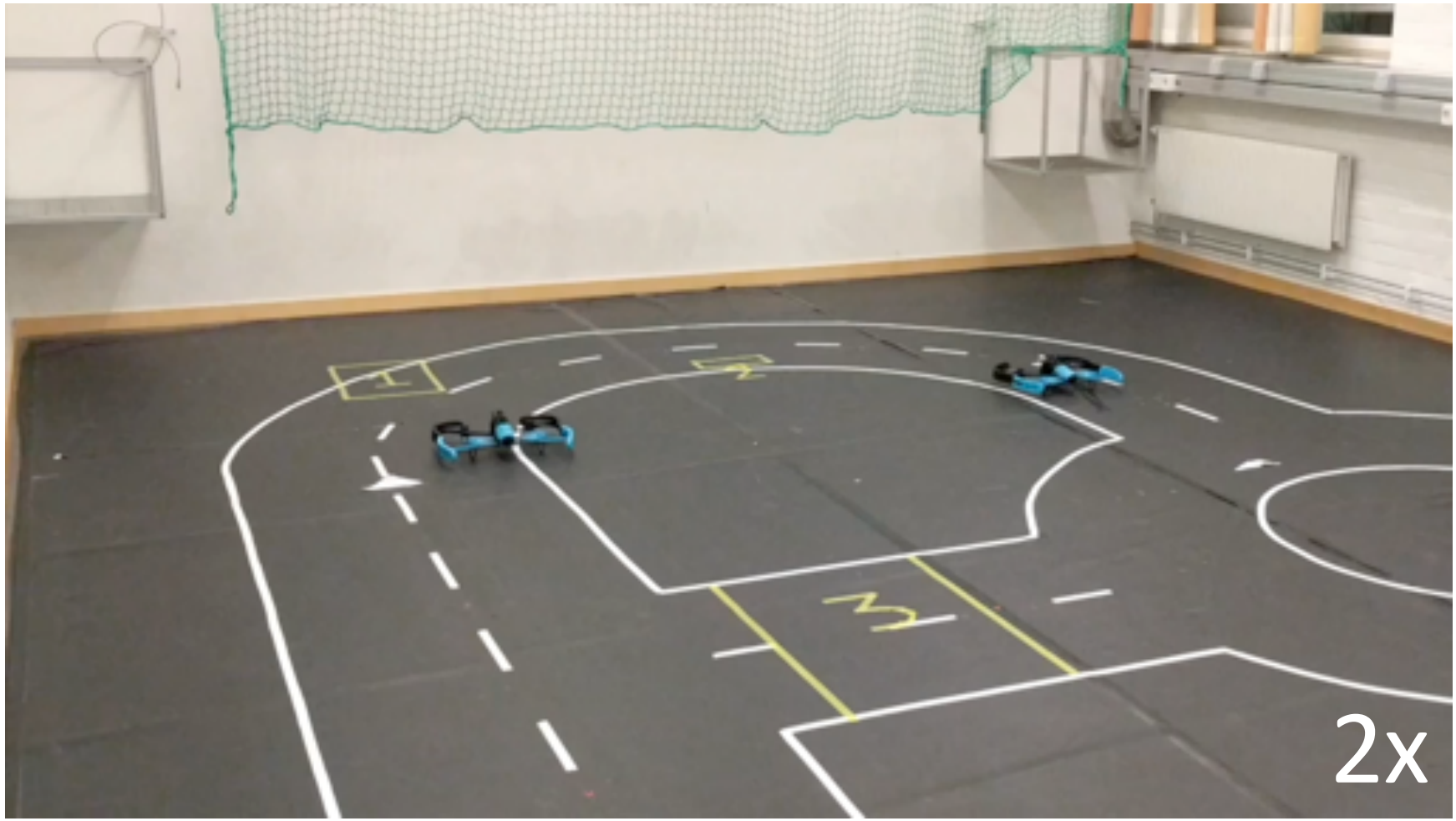


Cooperative Maneuvers





Cooperative Maneuvers



Quo Vadis?

Networks

- Internet of Things
- Software Defined Networks
- LoRA
- 5G
- Security, Dependability

Data & AI

- Edge AI
- Distributed AI
- (Deep) Reinforcement learning
- Distributed Learning & Decision Making

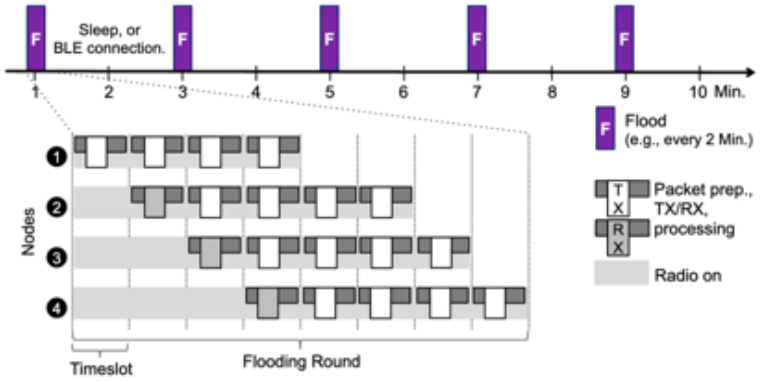
Applications

- Smart City
- Cooperative Driving
- Industry 4.0
- Marine Systems
- Green Energy

Master's Project – Improving BlueFlood

- **Internet of Things:** everything is connected and communicate over wireless
  - WiFi, Zigbee, Bluetooth
  - But energy is limited! We need efficient wireless communication

- **BlueFlood**
  - Protocol for concurrent transmissions / flooding on BLE
  - low-latency, energy-efficiency, high reliability
  - -> The goals of network protocols



Master's Project – Improving BlueFlood

- **Possible Tasks:**
  - Port and integrate BlueFlood into the Contiki-NG operating system
  - App integration of BlueFlood for advertisements
  - Blue-LWB: Low-Power Wireless Bus on top of BlueFlood

- **Requirements:**
  - Participation in the IoT course
  - Good knowledge of C
  - Motivation for low-level development (reading datasheets is part of the game!)

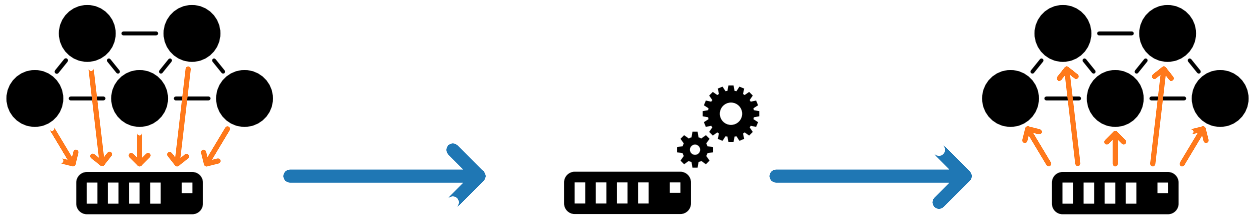
After the project?

Possibility to continue with a M.Sc thesis!

Examples:

- **Thesis 1:** BlueFlood at the 2021 dependability competition (international competition testing protocol's resiliency against *heavy* interference)
- **Thesis 2:** Advanced app integration topics: Developping an app using Blueflood, to have building-wide Bluetooth communication
- **Thesis 3:** Blue-LWB, a low-latency, energy efficient all-to-all communication primitive
- **Thesis 4:** Minimal BLE mesh on top of BlueFlood

Several Theses on Central Scheduling

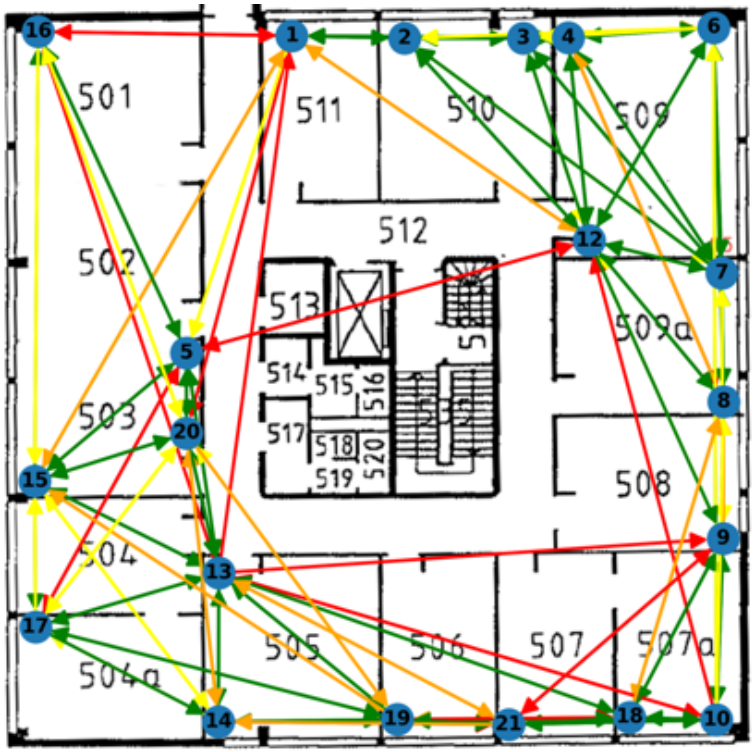


- **B.Sc:** Implementation and testing of deadline-based scheduling algorithms
- **B.Sc:** Stability analysis of our scheduler (C-TSCH) in different environments/conditions
- **B.Sc / M.Sc:** Neighbor data collection of runtime neighbor discovery data
- **B.Sc / M.Sc:** Development of a schedule distribution mechanism
- **M.Sc:** Porting the MAC Protocol Time-Slotted Channel Hopping (TSCH) to a new platform (nRF52840)

More Bachelor's Theses: Testbed

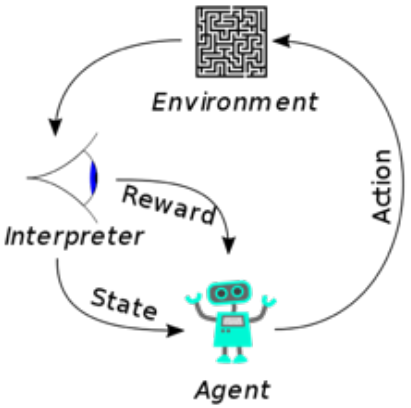
- We run wireless protocols on nodes scattered around the office

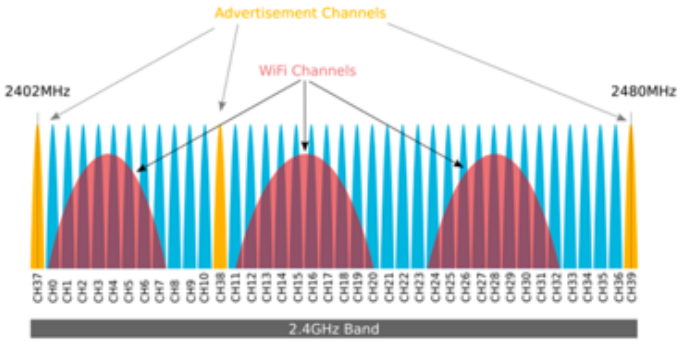
- **Thesis 1:** Development of a Job Scheduler for Wireless Communication Testbeds
- **Thesis 2:** Extend the testbed with a GPIO tracing capability to allow a deeper evaluation of new protocols



Master Thesis: Bluetooth: Adaptive Channel Blacklisting using Reinforcement Learning

- Bluetooth is everywhere
- Competes with WiFi for wireless access
- **Goal**: learn how to avoid collisions with WiFi and other Bluetooth devices by blacklisting channels
- **How**? Reinforcement Learning (the system tries again and again until it learned how to behave)





B.Sc Thesis: Testing Safe Intersection Crossing for Connected Vehicles

- Connected vehicles are (slowly) coming
- Soon, traffic lights and stop lights will be replaced by wireless communication!
- We have developed a protocol for intersection crossing
- C ho-based R c

Next Time

- Paxos
- Then
  - Recap
  - Project Presentations (Lab + Lecture)
  - AMA

Questions?

In part, inspired from / based on slides from

- Paul Francis
- Kenneth P. Birman
- Mike Freedman and Tanenbaum
- Hussam Abu-Libdeh
- Steve Schlosser
- Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels
- Jeff Dean
- Alex Moshchuk
- Vamsi Thummala, Prof. Cox