EE 579: Wireless and Mobile Networks Design & Laboratory

Lecture 11

Amitabha Ghosh
Department of Electrical Engineering
USC, Spring 2014

Lecture notes and course design based upon prior semesters taught by Bhaskar Krishnamachari and Murali Annavaram.

Outline

- Presentation and Demo Plans
- Recap

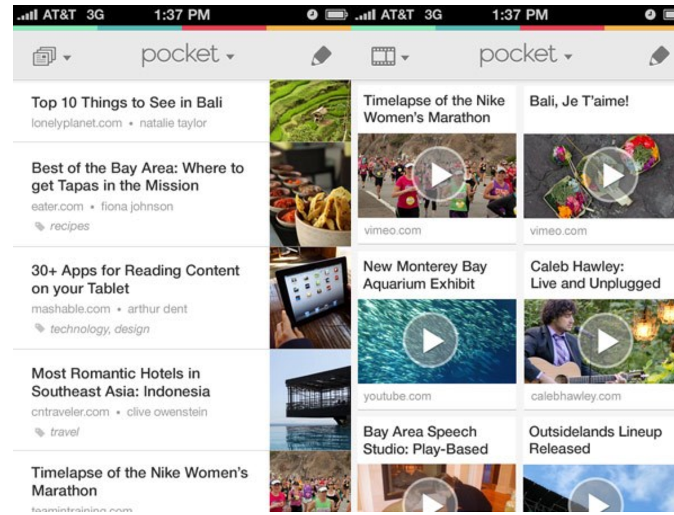
Smartphones, Platforms, Apps

- Design and develop network protocols and applications wireless and mobile devices

Smartphone Apps

The Good Enough Revolution: When Cheap and Simple Is Just Fine
Wired Magazine, Aug 24, 2009

- Pocket
  - ❑ When you find something interesting you want to view later, put it in pocket
  - ❑ iPhone, iPad, Browsers, …



- Car GPS
  - ❑ Entry level Garmin $80-$100
  - ❑ MotionX GPS Drive
  - ❑ 4.4 Stars, $0.99
  - ❑ Voice navigation costs

Smartphone Apps

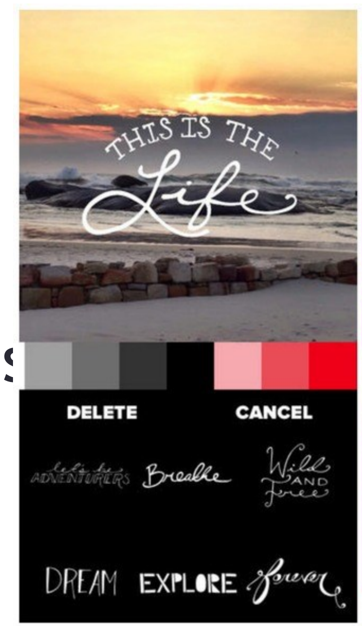
- Skype
  - Quality can be sketchy, but
  - Free or VERY low cost, 4+ rating
  - Available everywhere
  - The user experience is uniform

- Instagram
  - Social video-shooting and sharing
  - Jazz up photos and videos with filters
  - Change the focus while shooting
  - Free, 4+ rating

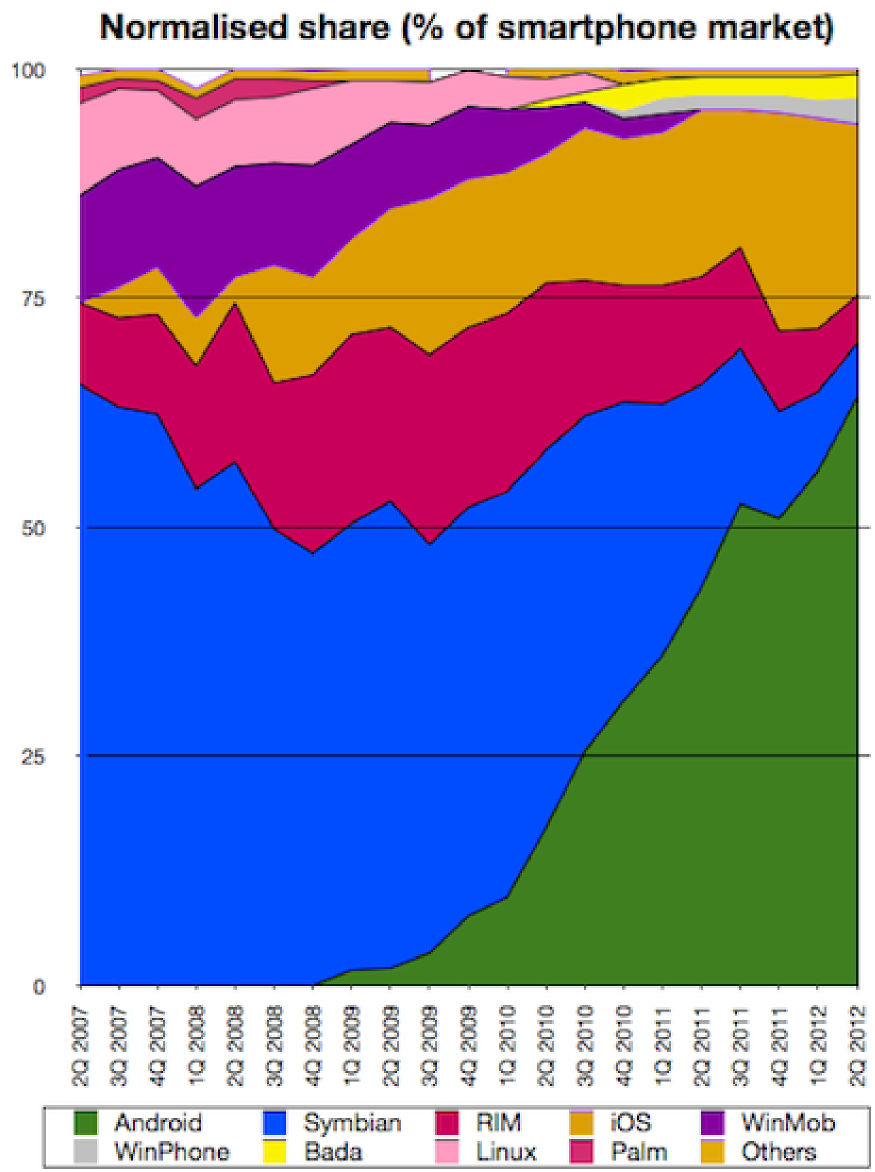
Smartphone Apps

- Duolingo
  - Apple’s top app of 2013
  - Learn a new language
  - Spanish, German, French, …
  - Both visual and verbal lessons
  - Free. Reward points to buy perks

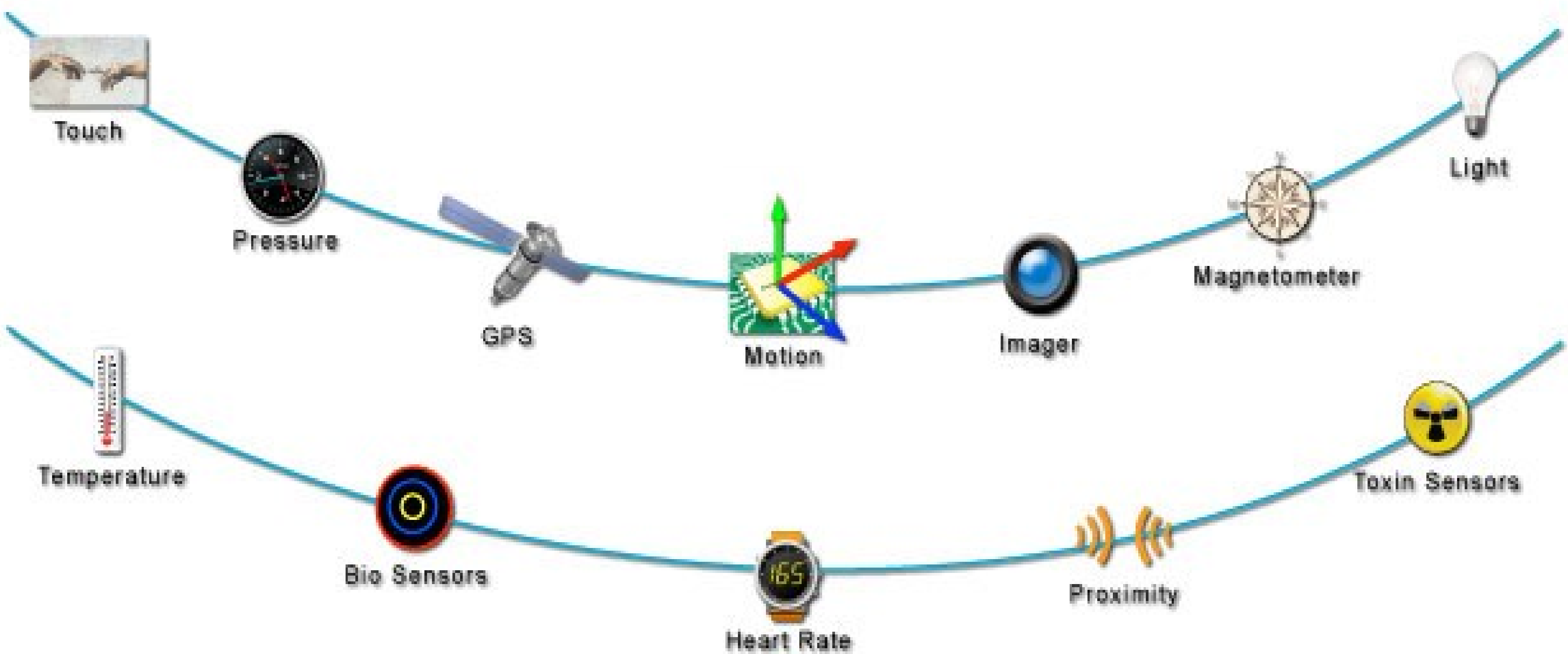
- Over
  - Overlay text on images
  - Turn routine pictures into e-cards
  - Price: $1.99



Market Share

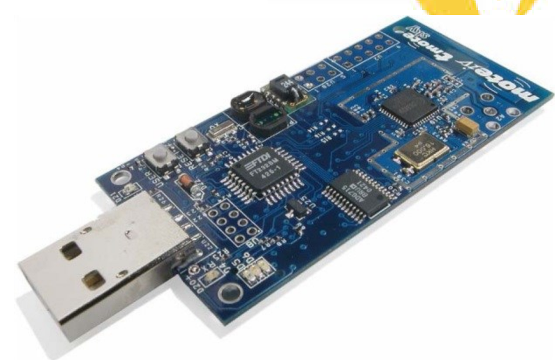


Sensors



Touch
Pressure
GPS
Motion
Imager
Magnetometer
Light
Temperature
Bio Sensors
Heart Rate
165
Proximity
Toxin Sensors

Wireless Sensor Platforms



- Contiki (v2.7 Nov 2013)
  - Open source OS for the Internet of Things
  - Connects tiny low-cost, low-power microcontrollers to the Internet
  - Written in C
  - Cooja simulator, emulated before burned into hardware
  - Supports IPv4, IPv6, 6lowpan, RPL, CoAP
  - Coffee flash file system
  - Protothreads – event-driven and multi-threaded
  - Runs on a range of low-power wireless devices
  - ContikiMAC sleepy routers
  - Atmel, Cisco, ETH, Redwire, SAP, Thingsquare

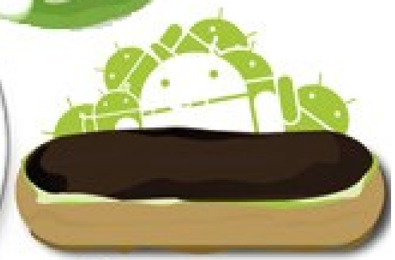
Google acquires Android Inc. in 2005, **Android 1.0 Astro**, Sept 2008

Android 1.5 Cupcake, April 2009, 1st commercially available version with Android's first touch-screen phone HTC Magic

Android 1.6 Donut, Sept 2009, text-to-speech technology, search by text and voice

Android 4.1 Jellybean, July 2012, Google Now, faster smoother more responsive

Android 2.0/2.1 Eclair, Oct 2009, live wallpapers, virtual keyboard, Bluetooth, HTML5, improved navigation with Google maps



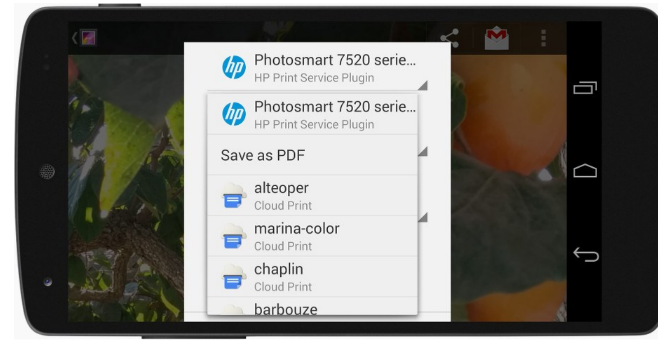
Android 4.0 Ice Cream Sandwich, Oct 2011, performance and speed, tablet features on smartphones, GTalk

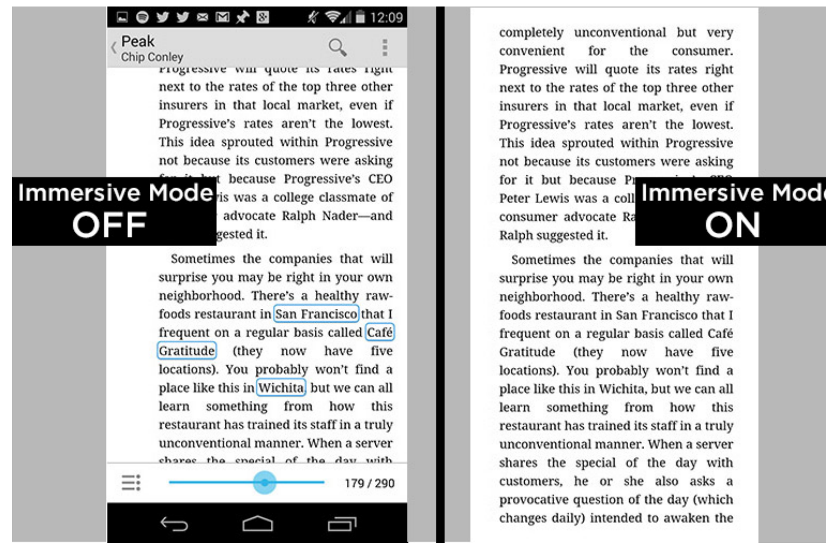
Android 2.2 Froyo, May 2010, OS speed with Java V8 engine and JIT compiler, Flash, remote wipe features

Android 3.0 Honeycomb, Feb 2011, designed for tablets, no need for physical buttons, system bar, action bar, redesigned keyboard

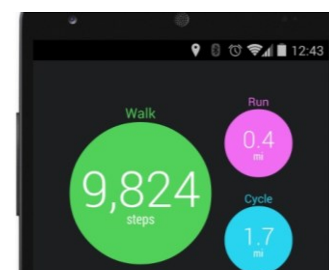
Android 2.3 Gingerbread, Dec 2010, quick front and back camera switch, better battery mgmt, near field communication (NFC) with Google Wallet

Nov 2013 (Latest)





- Fast and smooth on a range of devices, millions of entry-level devices < 512 MB RAM
- Printing over Wi-Fi or cloud
- Full-screen immersive mode (use every pixel, capture touch events)
- Secure NFC through Host Card Emulation (HCE)
- Low-power sensors (e.g., step detector and counter)



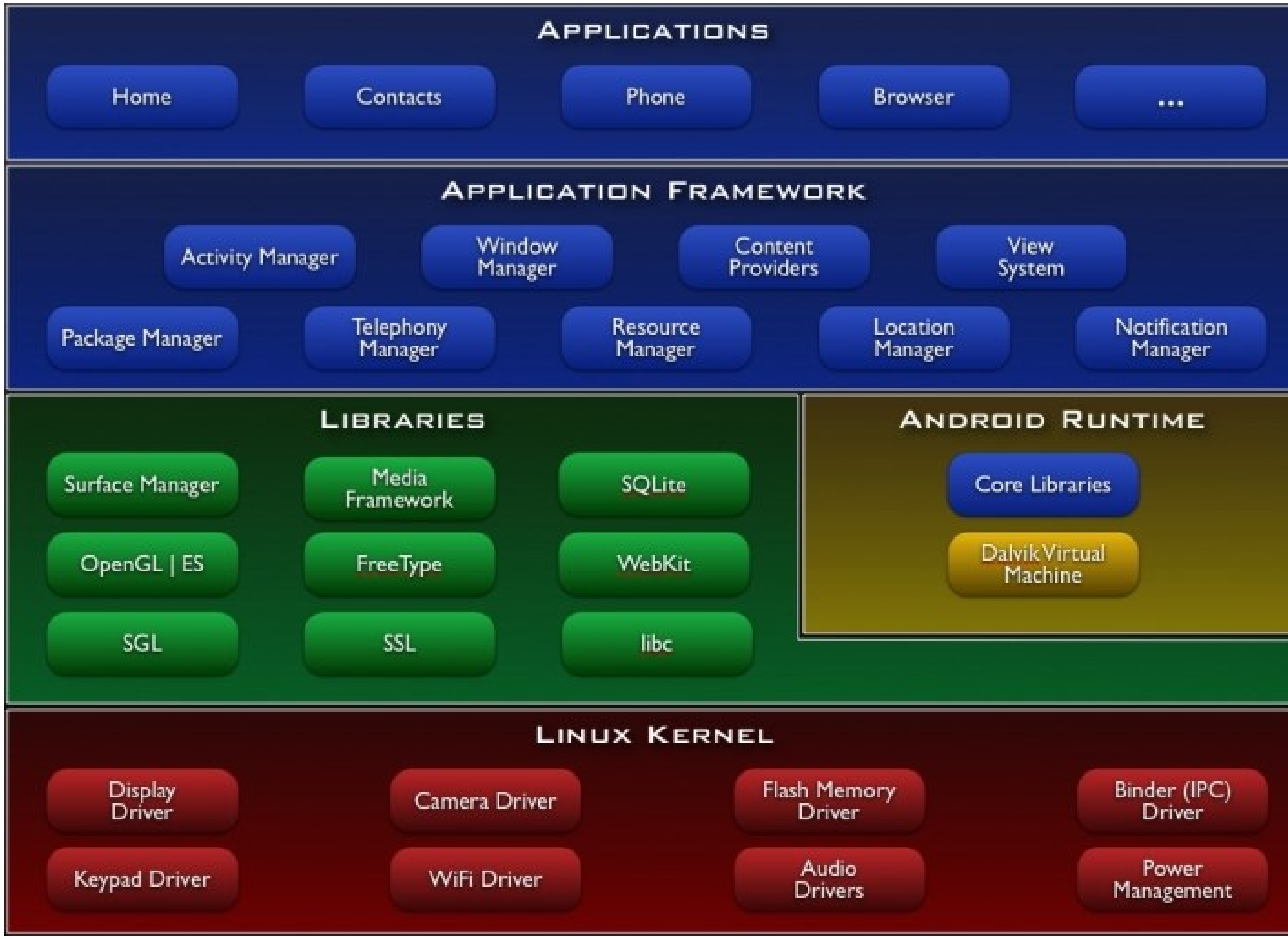
Android Architecture

Written in Java, executed in Dalvik VM. Home, Contacts, Phone, Browser, …

Written mostly in Java. Managers for Activity, Window, Package, …

Native libraries, daemons and services (C/C++). SQLite, OpenGL, SSL, … Dalvik VM, Core libs

Drivers for hardware, networking, file system access, and inter-process-communication (IPC). Display, camera, flash, Wi-Fi, audio, …



The Linux kernel, the libraries, and the runtime are encapsulated by the Application Framework. Developers typically work with the top two layers

Android Runtime

- Core Java Libraries
- Dalvik Virtual Machine (Dan Bornstein from Google)

Dalvik Virtual Machine

It is the software that executes Android apps (not the Java VM), specifically designed to run on

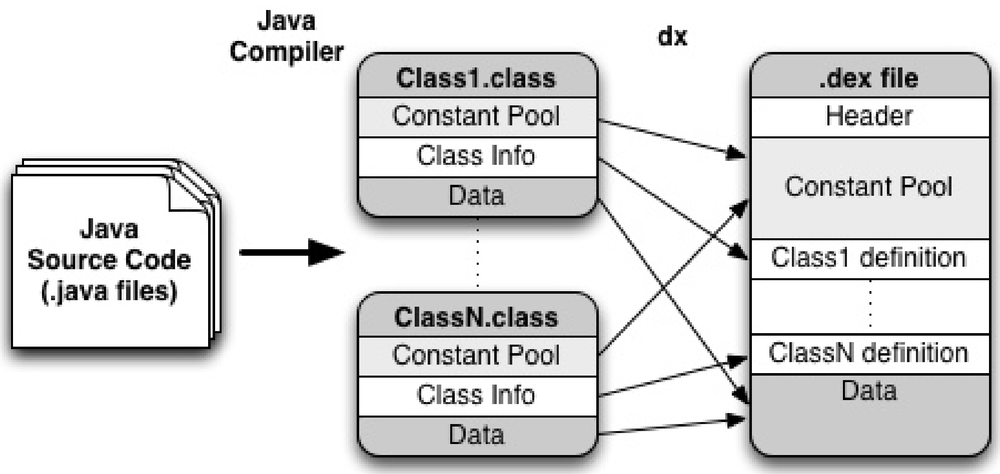
- Slow CPU
- Relatively little RAM
- OS without swap space
- Powered by a battery
- Diverse set of devices
- Sandboxed application runtime for security, performance, and reliability

Somewhat conflicting constraints

Dalvik Virtual Machine

Typical Workflow

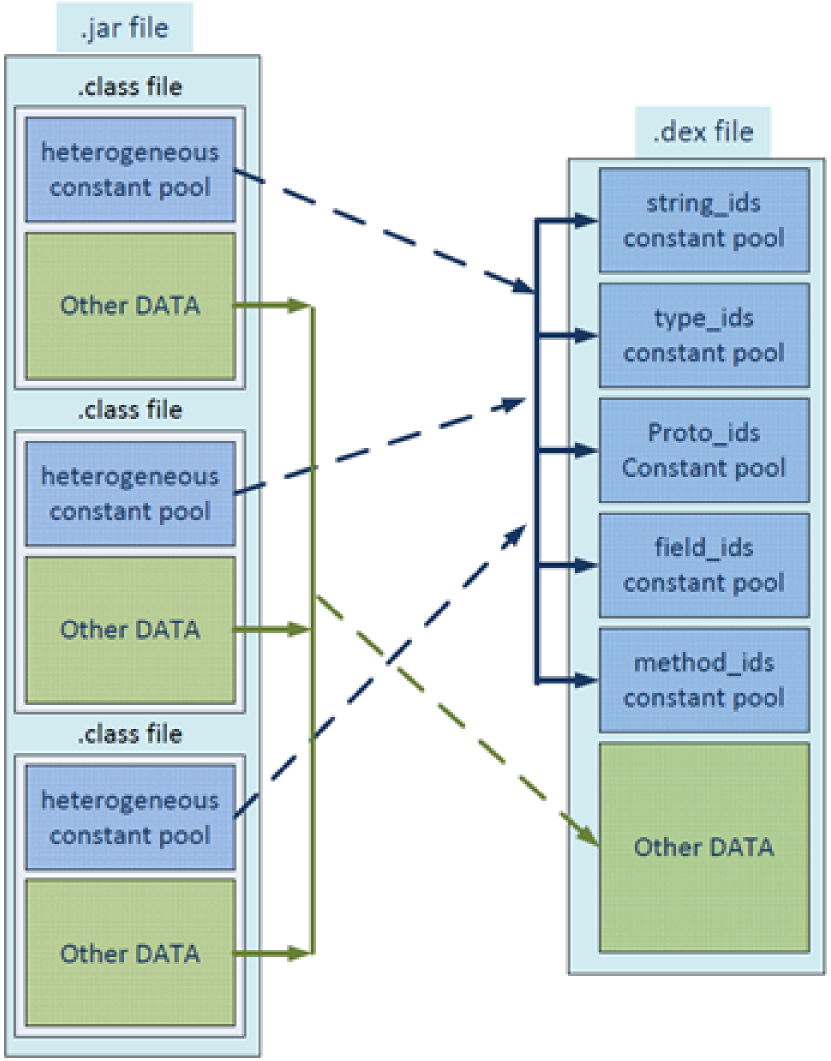
- Write apps in Java
- Compile into Java bytecode
  - One .class file per class
- DX tool converts multiple Java classes into a single DEX file (classes.dex)
  - Rearranges classes, remove redundancy
- Dex file is packaged with other resources and installed on device



Dalvik Virtual Machine

Conserving Memory

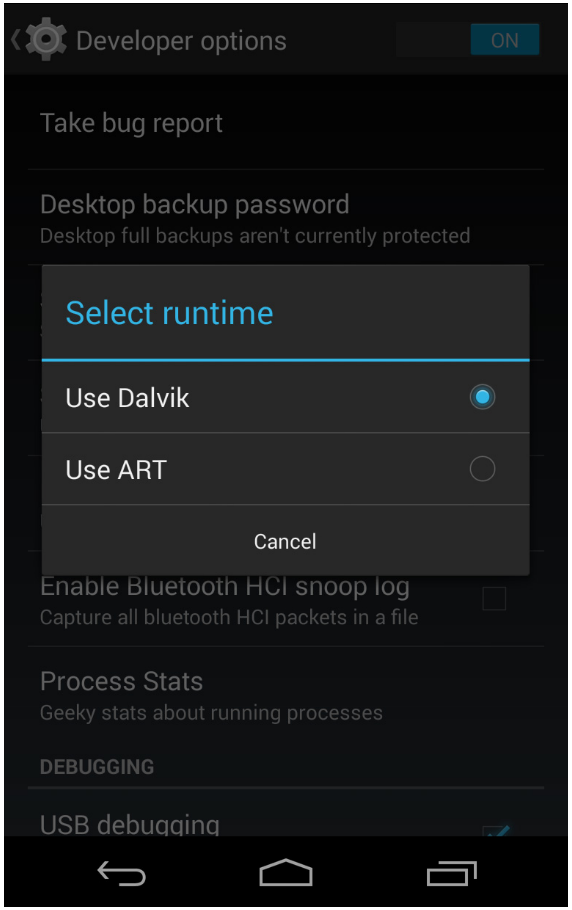
- .dex uses shared, type-specific constant pools
  - Minimal repetition and more logical pointers than a .class file
- A constant pool stores all literal constant values within the class
  - String constant, field, variable, class, interface, and method names
- In a .class file, constant part: 60%, method part: 33%



Android ART

Android Run Time : Google finally moves to replace Dalvik, to boost performance and battery life. Early version included in Android KitKat

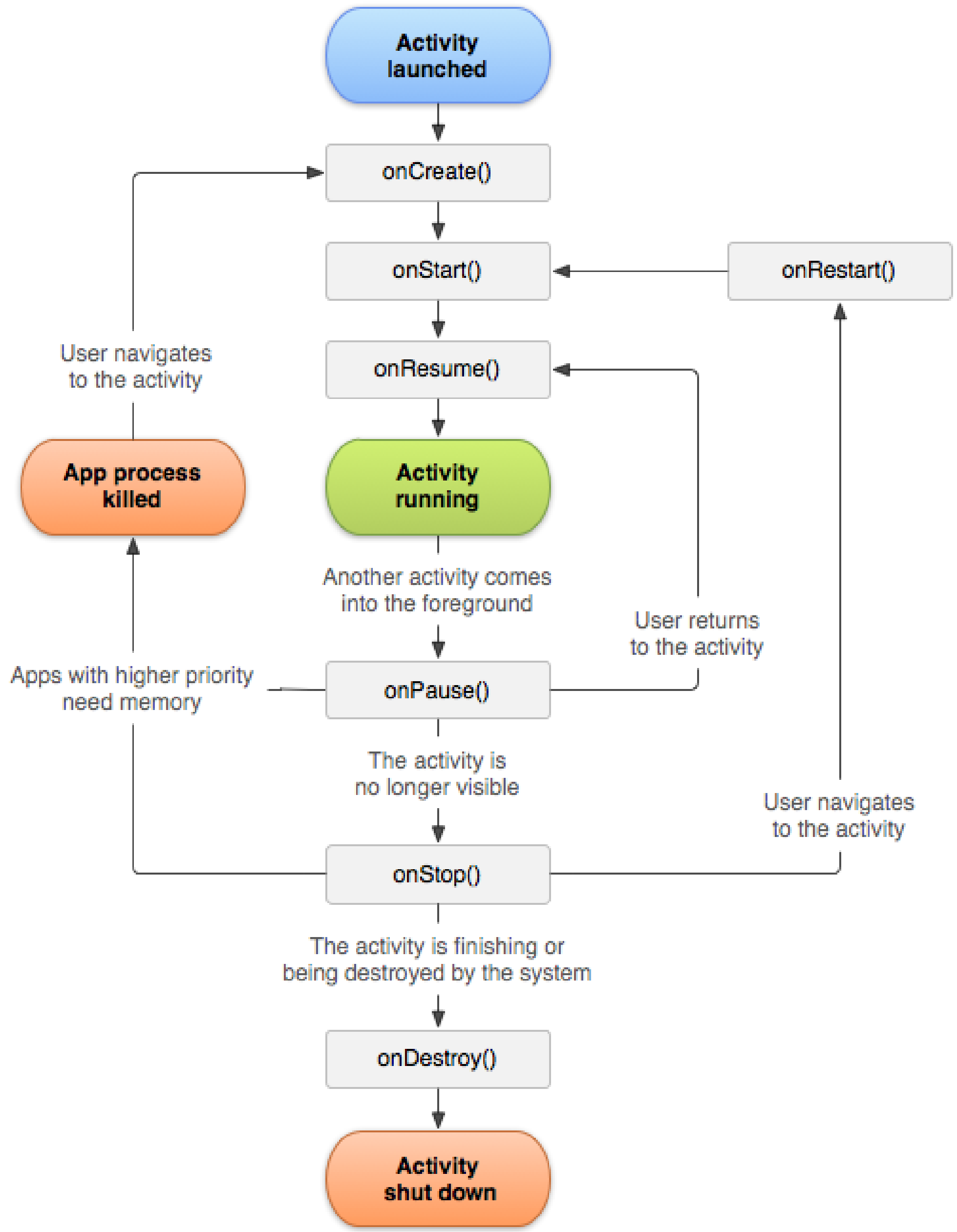
- ART straddles a middle-ground between compiled and interpreted code, called “ahead-of-time” (AOT) compilation
- Currently apps are interpreted at runtime using JIT (slow), compare with iOS
- With ART, app is compiled into native code while installing (fast)



App Lifecycle

Lifecycle is a set of states

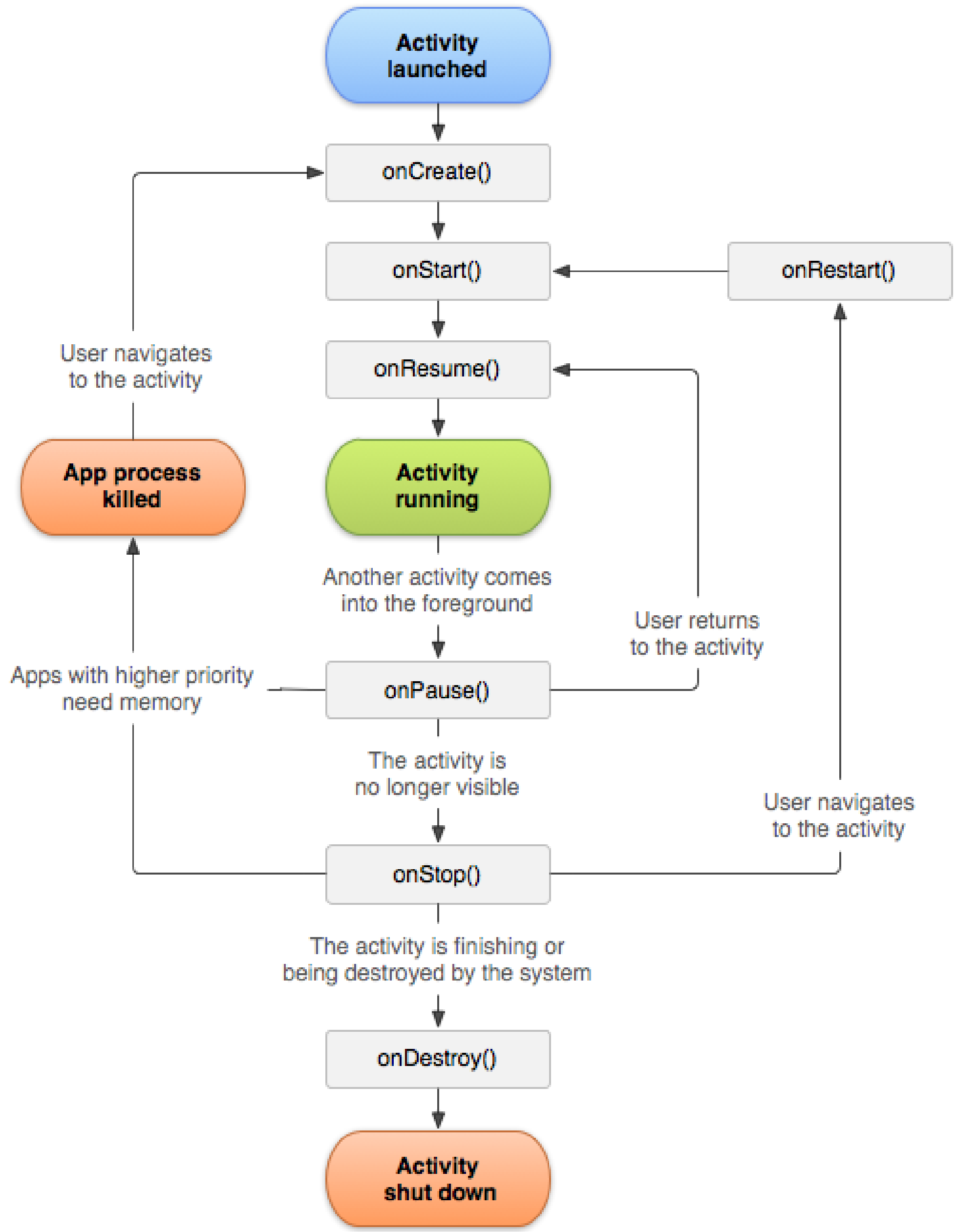
- When the current state changes, Android OS notifies the Activity of that change
- Implemented by callback methods



App Lifecycle

Four States

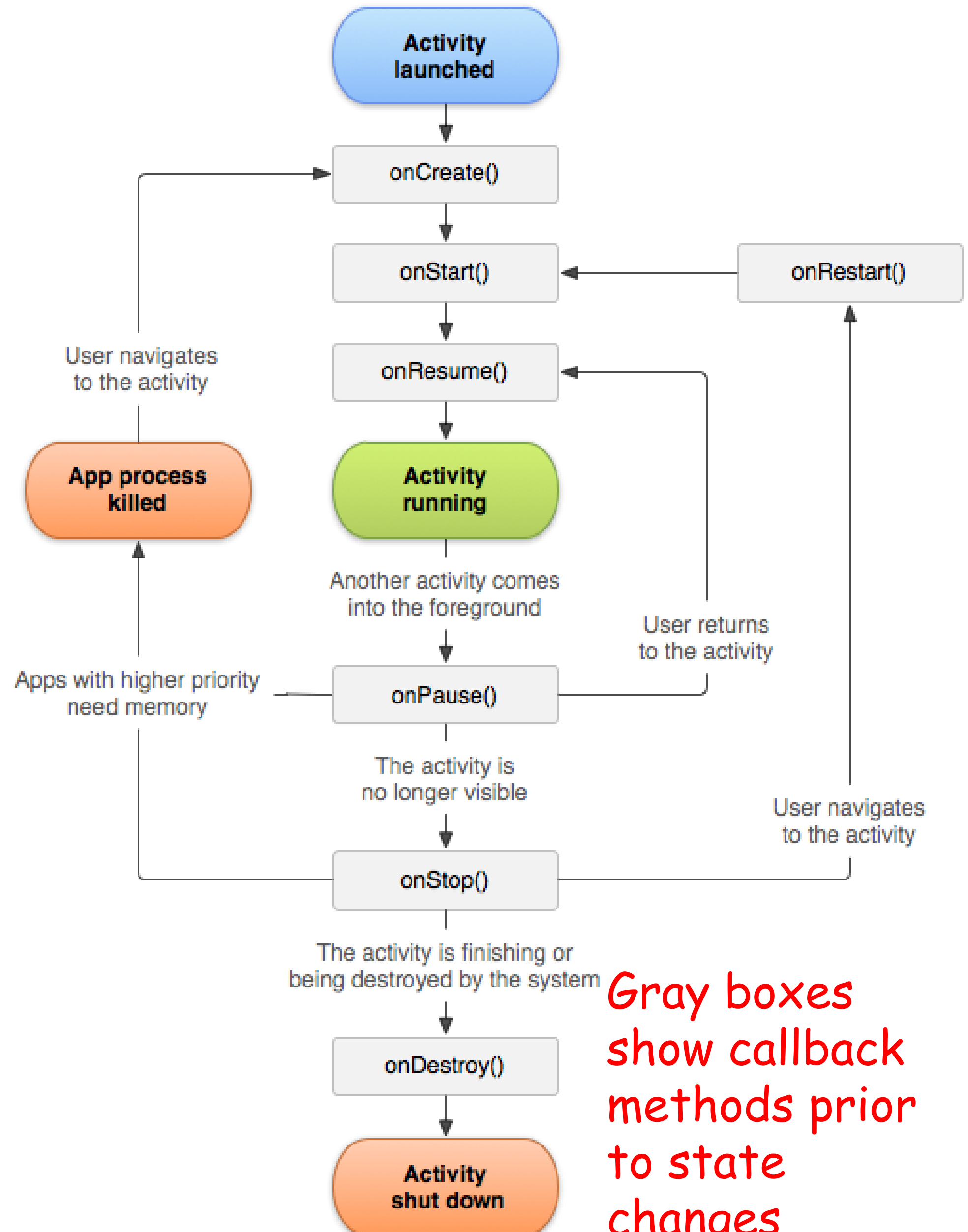
- Active / Running
  - Visible, has focus, and in foreground
- Paused
  - Partially visible but not active and lost focus
  - Completely alive and maintains its state
- Stopped
  - Completely obscured by another activity
- Destroyed / Dead
  - No longer in memory



App Lifecycle

Seven Callback Methods

- onCreate() – UI creation and initialization of data elements
- onStart() – called before Activity is visible (but not alive)
- onResume() – Activity becomes visible and active for user to interact
- onPause() – another Activity comes in front, or user navigates away

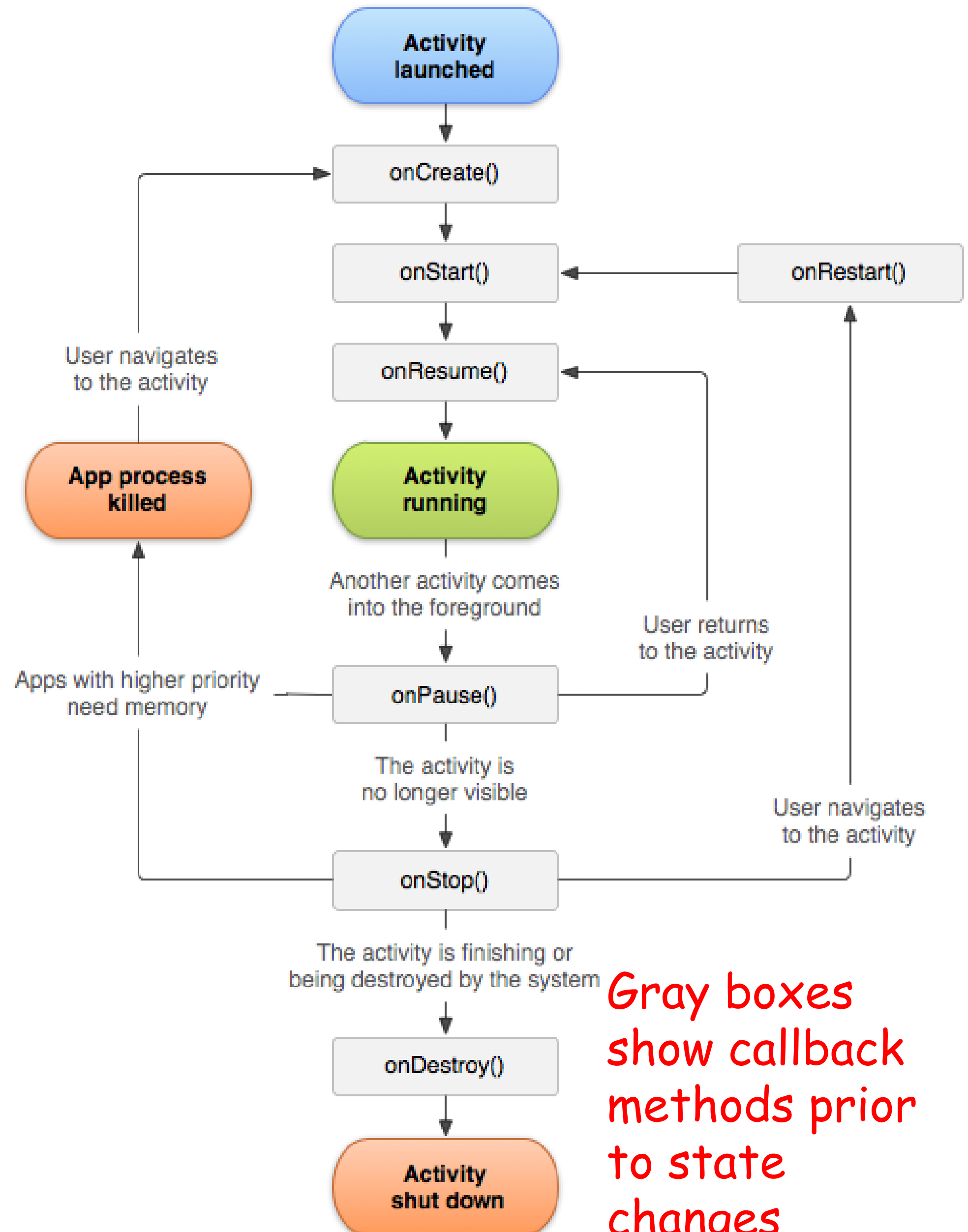


Gray boxes show callback methods prior to state changes

App Lifecycle

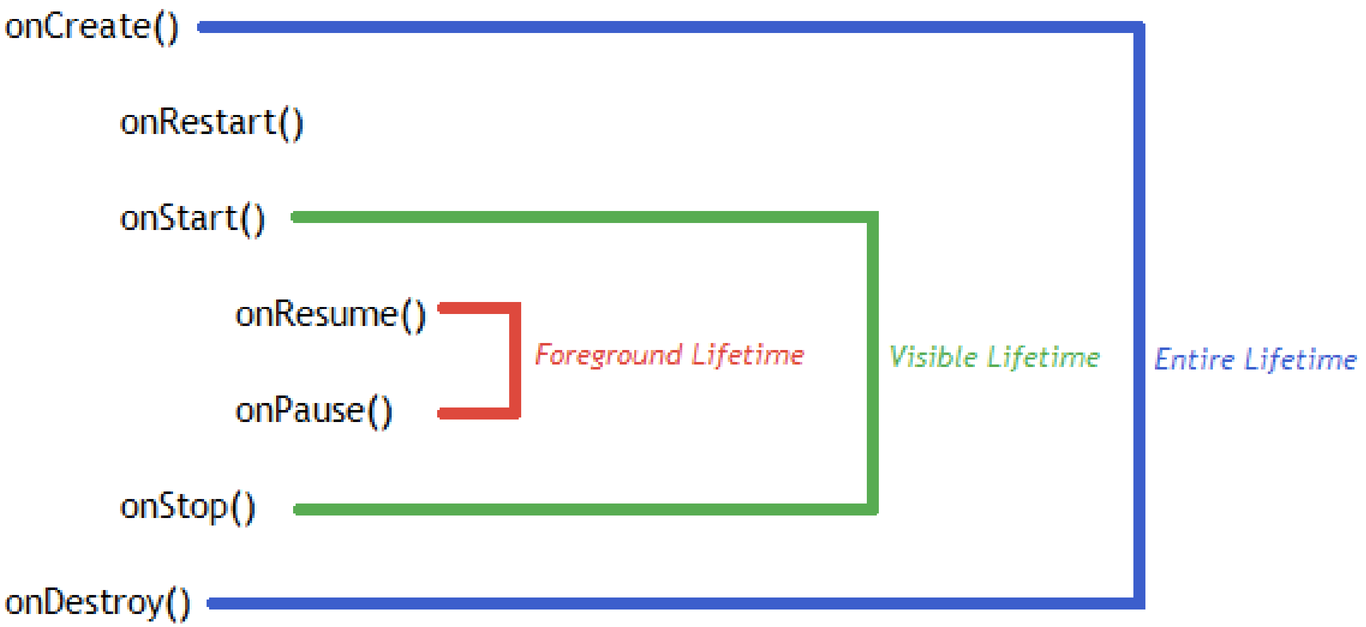
Seven Callback Methods

- onStop() – back button, or new Activity completely covers
- onRestart() – user navigates back to the Activity
- onDestroy() – Activity is destroyed



Gray boxes show callback methods prior to state changes

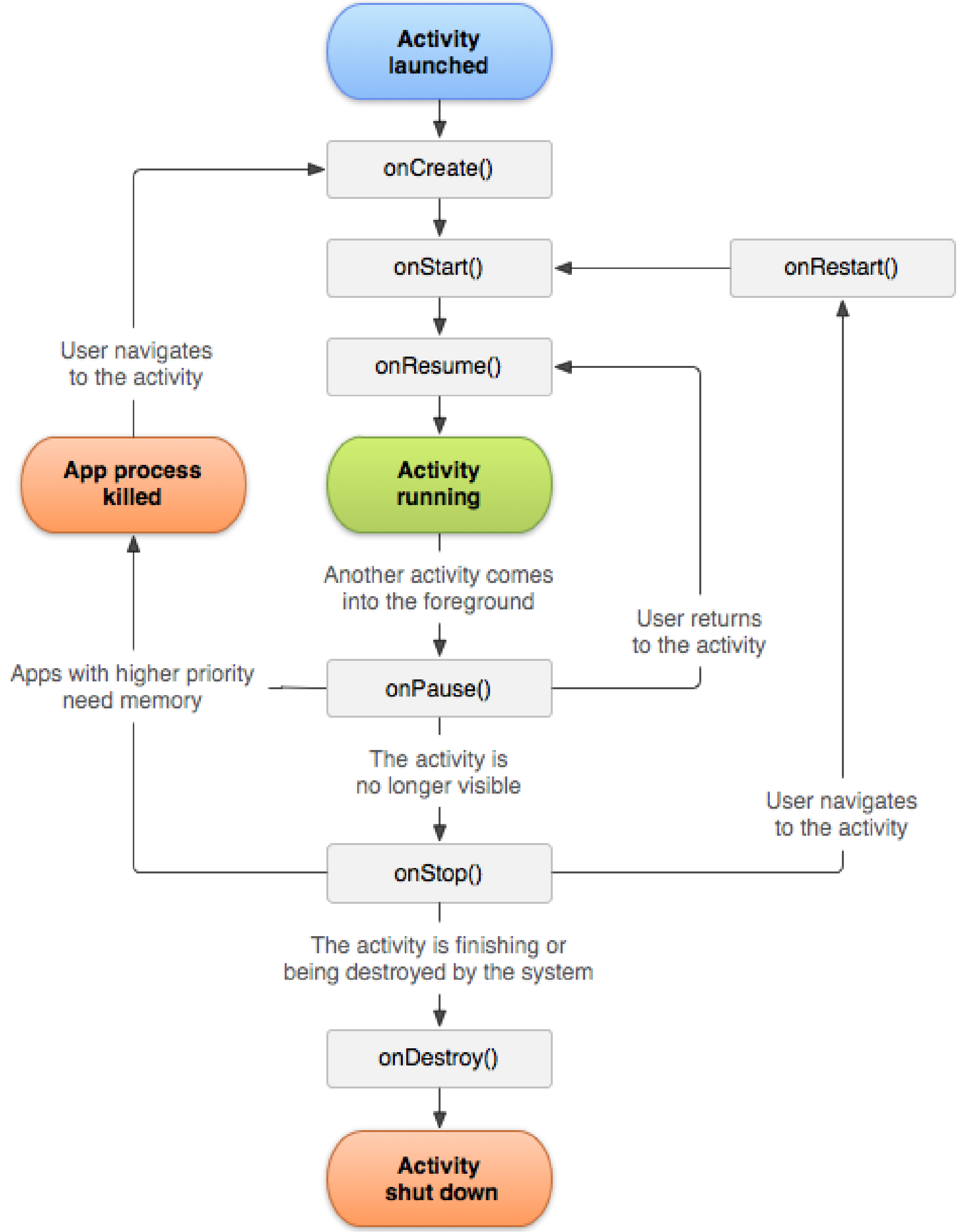
App Lifecycle



App Lifecycle

Three Lifecycle loops for every Activity, defined by callback methods

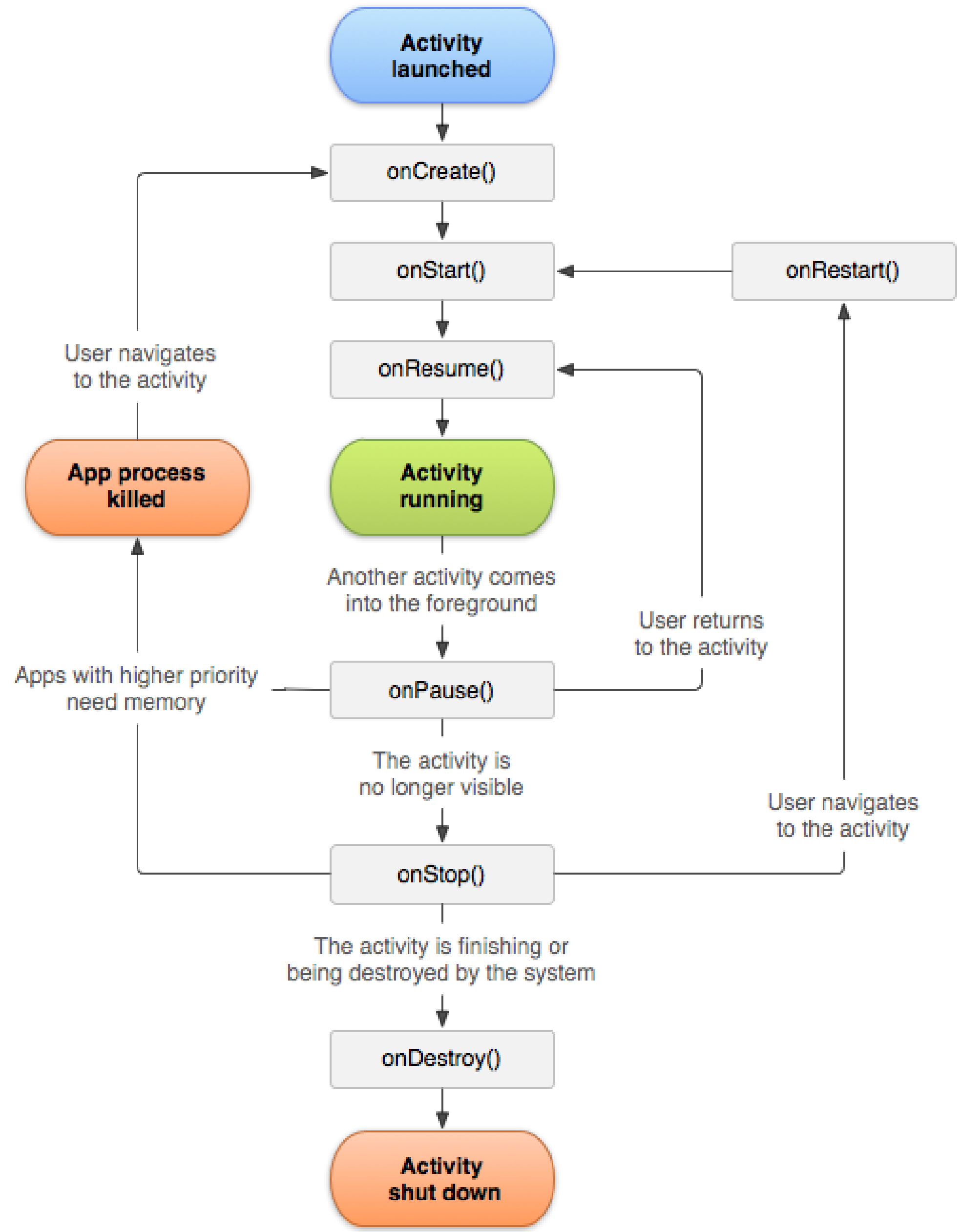
- Entire Lifetime – first call to onCreate() and final call to onDestroy()
- Visible Lifetime – from onStart() and onStop()
- Foreground Lifetime – from onResume() to onPause



App Lifecycle

Saving Persistent State

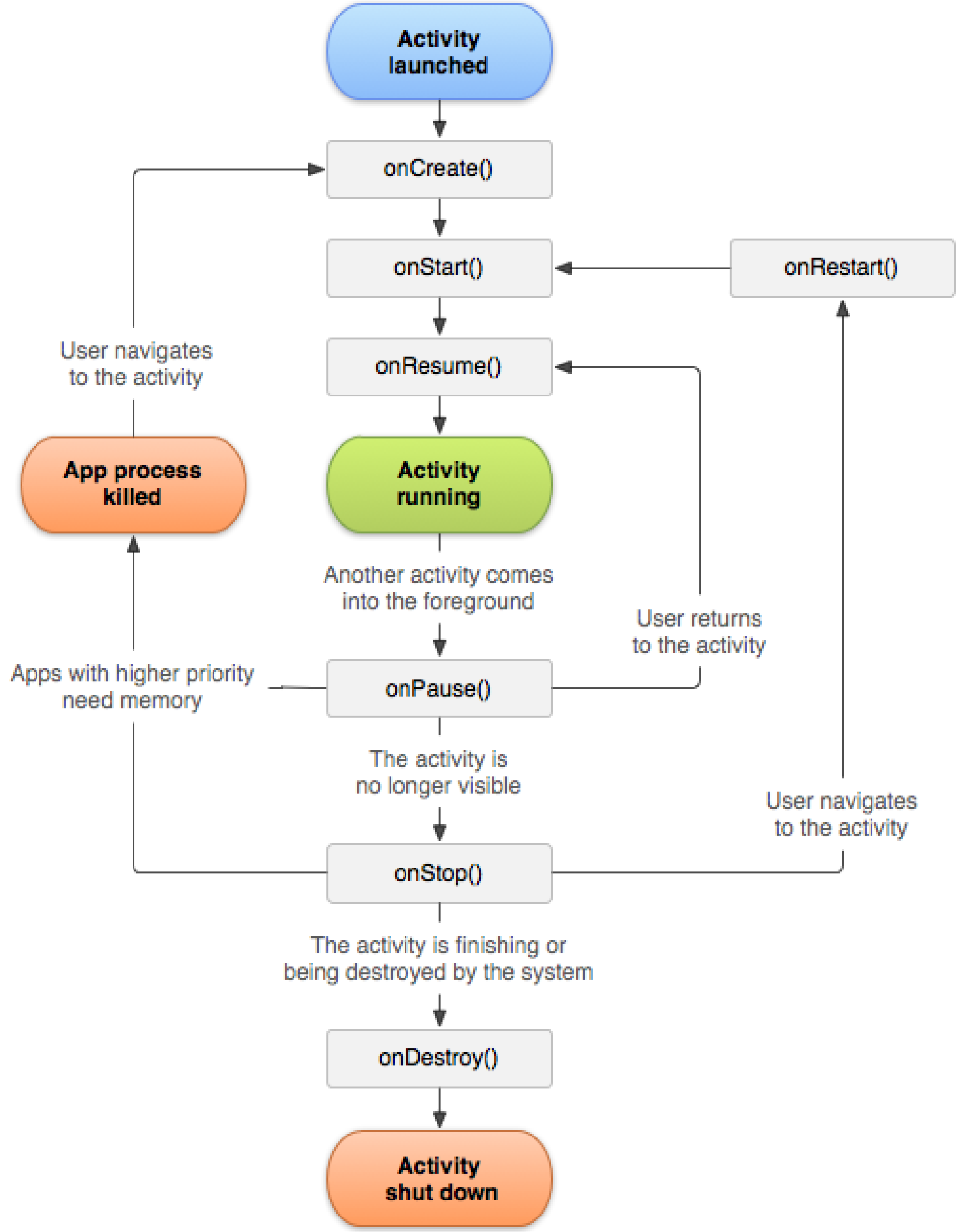
- When an Activity is stopped or paused, its state is preserved
- When an Activity is destroyed by the system, it is recreated next time Activity starts
- User is often unaware that an Activity is destroyed, resulting in surprises and crashes



App Lifecycle

Two Kinds of Persistent States

- Shared document-like data
  - SQLite storage using a content provider
  - “Edit-in-Place” user model
  - Backup fully at onPause()
- Internal state (user prefs)
  - API calls to store prefs
  - E.g., user’s initial calendar display (day vs week view), or default webpage in a browser



Fun with Math

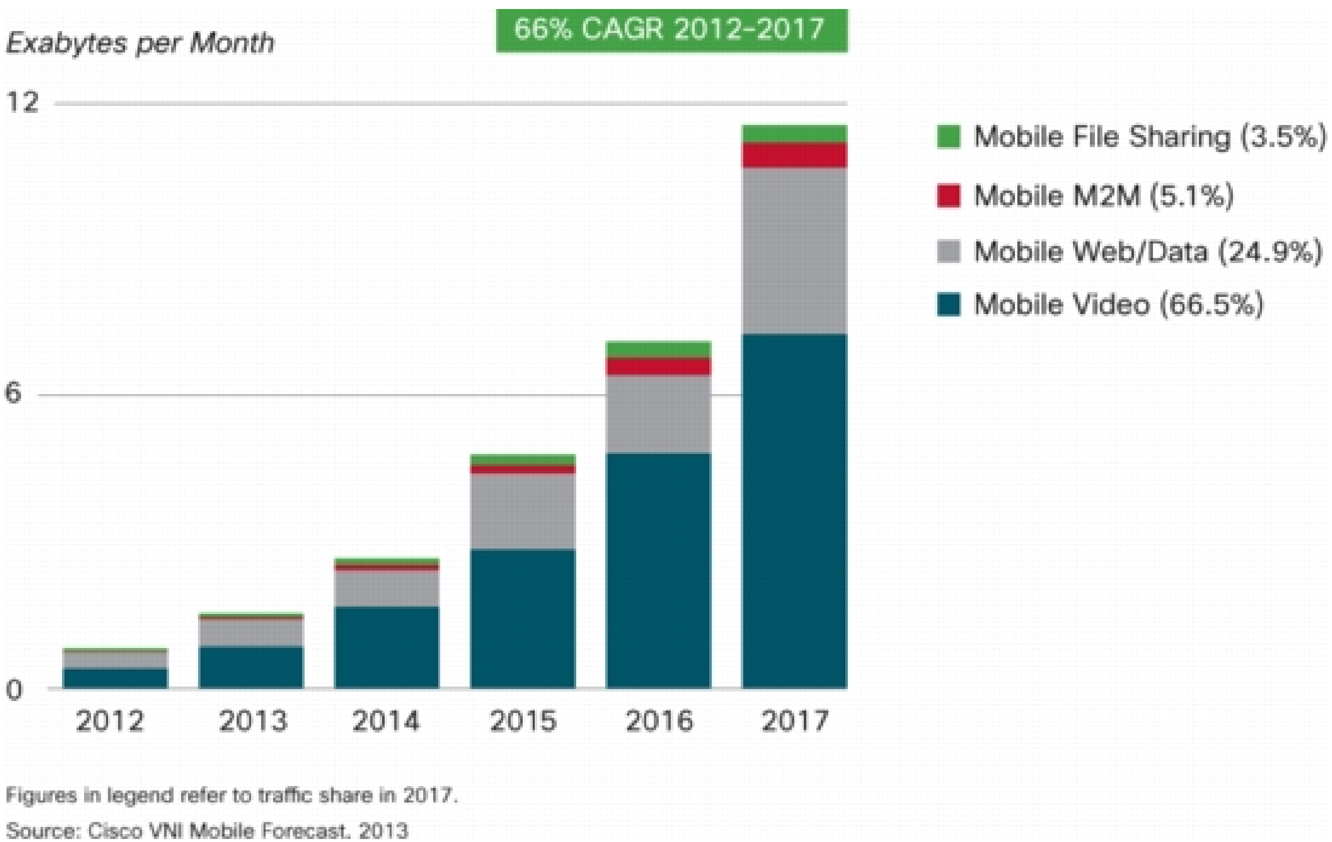
$$S = 1 + 2 + 3 + 4 + \dots ?$$

a) Infinity
b) Does not converge (diverges)
c) A finite value
d) A Googolplex $10^{(10)^{\wedge}100}$
e) Confused

$S = -1/12$ Is it Absurd?

Motivation

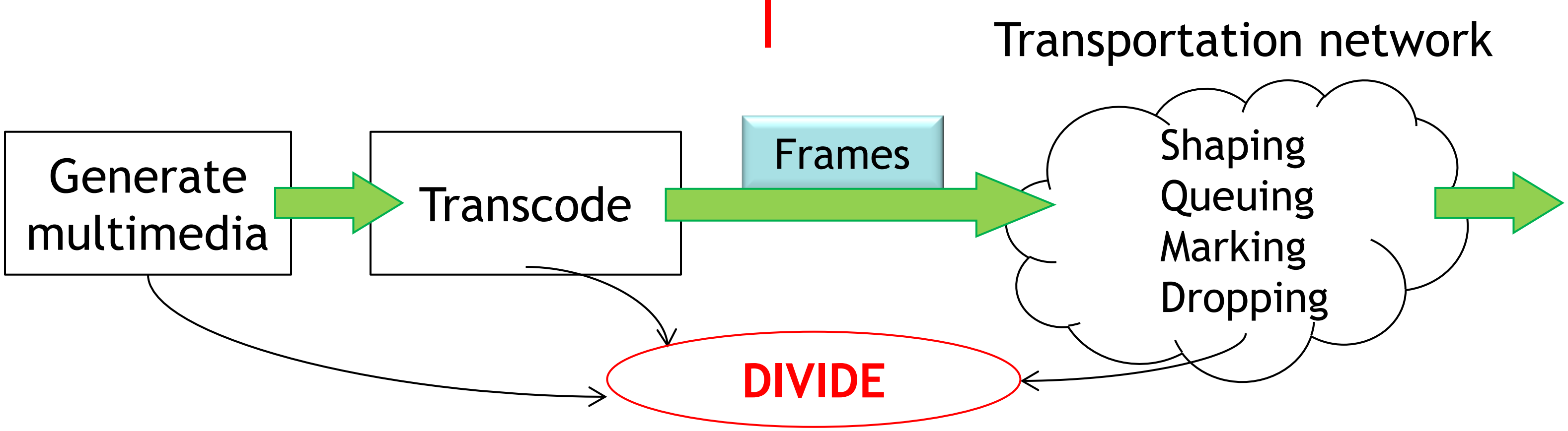
- Mobile Video Traffic Projection
  - Over 66% of all mobile data traffic will be from video by 2017
  - 7.4 exabytes (EB) out of 11.2 EB (1 EB = 10^{18} bytes)



Content-Pipe Divide

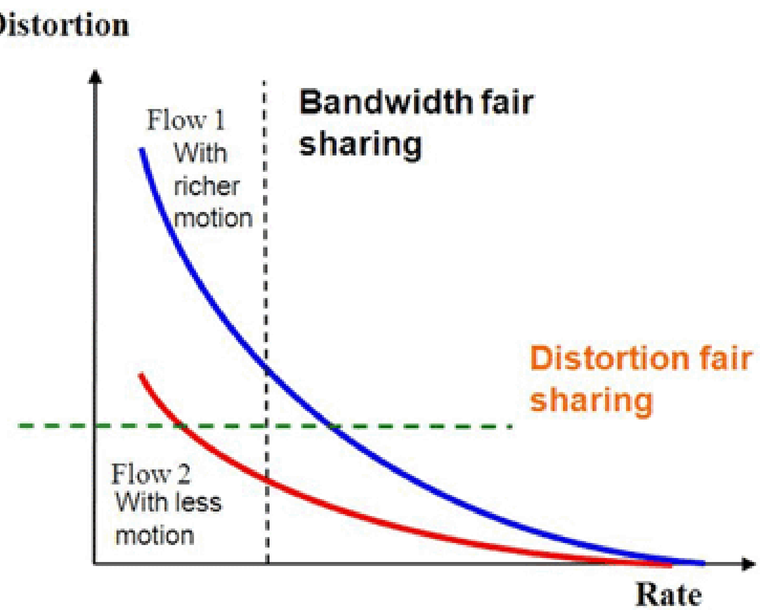
- Content Providers
  - Media companies, end-users, operators of CDN and P2P
  - Generate content treating the network as simply a means for communication (dumb pipes)

- Pipe Providers
  - ISPs, equipment & network management vendors, municipalities
  - Treat every content equally as simply bits to be transported between nodes (dumb content)



Content Aware Networking

- Protocol Fairness
  - Rate fair: Each flow gets half the capacity
  - Rate-Distortion fair: Flow1 gets more



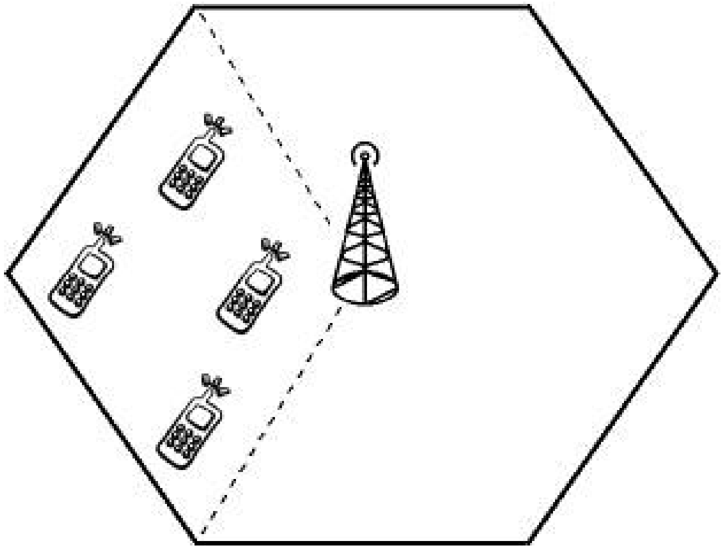
- A New Protocol Design Paradigm
  - Utilize content characteristics
  - Allocate resources based on the optimality criteria that are reflective of the content
  - More adaptive and effective network protocols that are rate-distortion fair

Content Aware Video Delivery over 3G WCDMA Networks

Kartik Pandit, Amitabha Ghosh, Dipak Ghosal, and Mung Chiang, "Content Aware Optimization for Video Delivery over WCDMA," *EURASIP Journal on Wireless Communications and Networking*, July 2012.
URL: http://anrg.usc.edu/~amitabhg/papers/EURASIP-2012.pdf

Network Model

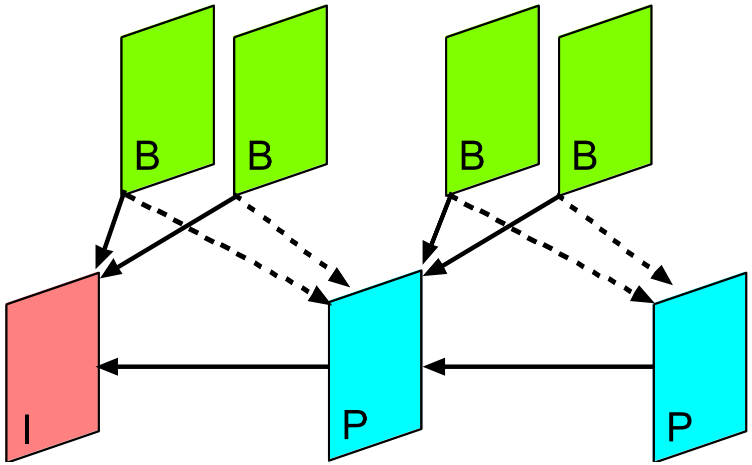
- Cellular Uplink
  - Increasing demand for high data rate
    - EVDO RA (1.8 Mbps), LTE (50 Mbps)
  - A single WCDMA cell, with a base station serving all users
  - Each user transmits a pre-encoded video upstream
  - Videos are encoded as GOP (Group of Pictures) structures



- Degrees of Freedom – Control
  - Scheduling (send or drop frame)
  - Transmission power

Video Model

- Group of Pictures (GOP)
  - Successive frames organized into a repetitive structure
    - I frame (intra) – coded independently
    - P frame (predictive) – motion-compensated difference, depends on previous P frame
    - B frame (bipredictive) – depends on previous and following P/I frames
  - Idea: Drop unimportant frames without hurting the quality



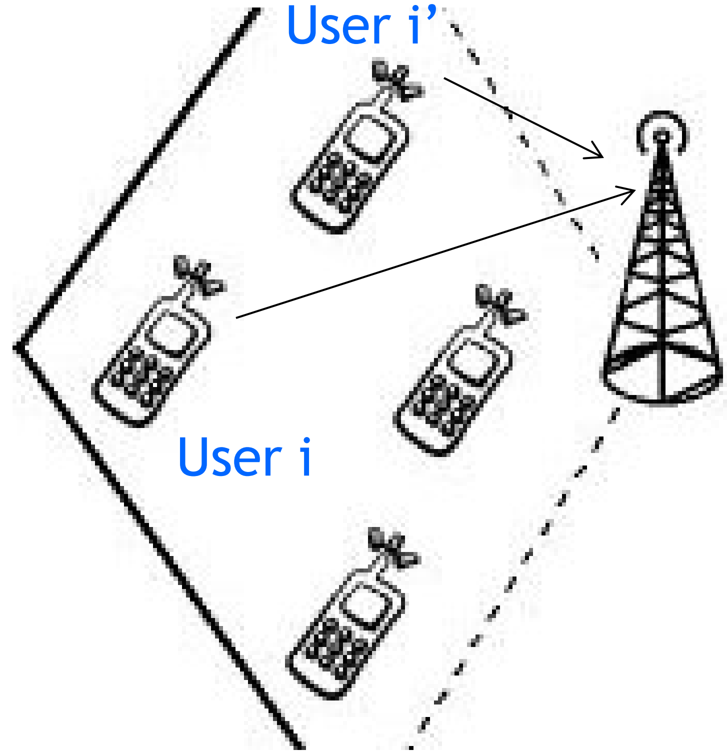
GOP: IPBBPBB
Directed acyclic graph
Arrows indicate dependency

Constraints

- SINR – signal to interference plus noise ratio

- Achievable rate

$$\sum_{j=1}^{n} d \log\left(1 + c\gamma_{ij}\right)$$



$$\gamma_{ij} = \frac{p_{ij} g_{ii} (1 - \theta_{ij})}{\sum_{i'=1, i' \neq i}^{N} p_{i'j'} g_{i'i} (1 - \theta_{i'j'}) + \eta_0}$$

Optimization Formulation

- Content-Aware Distortion-Fair Optimization (CADF)
  - Minimize the sum of distortions over a GOP for all videos subject to SINR constraints

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{N} D_i(\Lambda_i) \\
\text{subject to} \quad & R_i\left(\Lambda_i\right) \leq \sum_{j=1}^{n} d \log\left(1 + c\gamma_{ij}\right), \quad \forall i \\
\text{variables} \quad & 0 \leq p_{ij} \leq p_{max}, \quad \forall i, \quad \forall j \\
& \theta_{ij} \in \{0, 1\}
\end{aligned}
$$

  - An NP-hard problem (MINLP)
  - Can solve efficiently using heuristics

QAVA: Quota Aware Video Adaptation

Jiasi Chen, Amitabha Ghosh, Josphat Magutt, and Mung Chiang, "QAVA: Quota Aware Video Adaptation," ACM CoNEXT, pp. 121--132, Nice, France, December 2012.

URL: http://anrg.usc.edu/~amitabhg/papers/CoNEXT-2012.pdf

Motivation: The Conflict

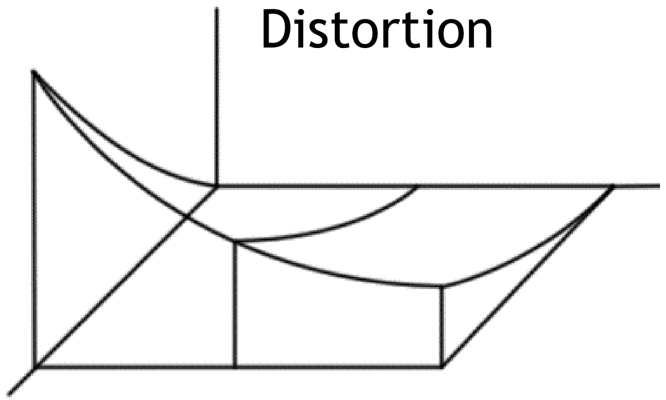
- Emerging Trends
  - Video traffic becoming dominant (>66% by 2017)
  - Usage-based pricing becoming prevalent
    - AT&T wireless (Jan 2012): $30/$50 for 3/5 GB (baseline) + $10 per GB
    - Verizon Wireless (July 2011): $30/$50/$80 for 2/5/10 GB (baseline) + $10 per GB

- Can the user consume more content without worrying about the wallet?

- Is every bit needed for everyone at all times?

QAVA: Graceful Tunable Tradeoff

A 3-way tradeoff



Within budget

Cost

Size of the video (bit-rate)

Minimize

Distortion

Video compressibility

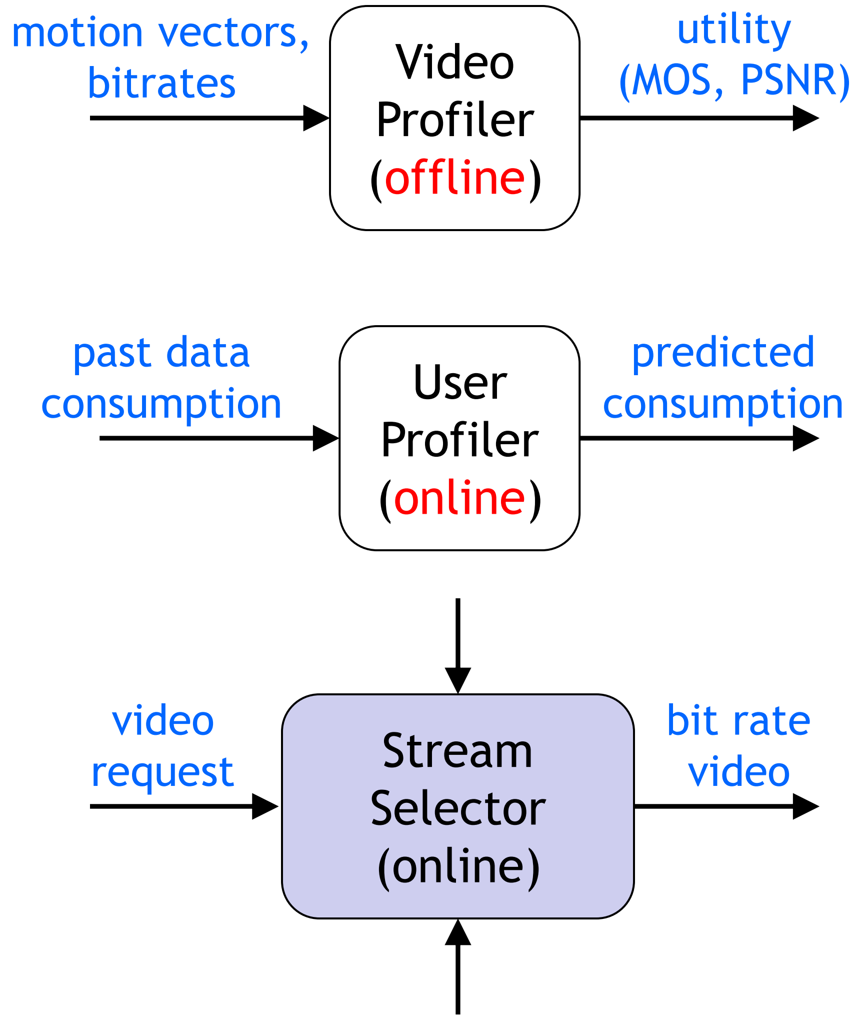
Supply

Videos watched

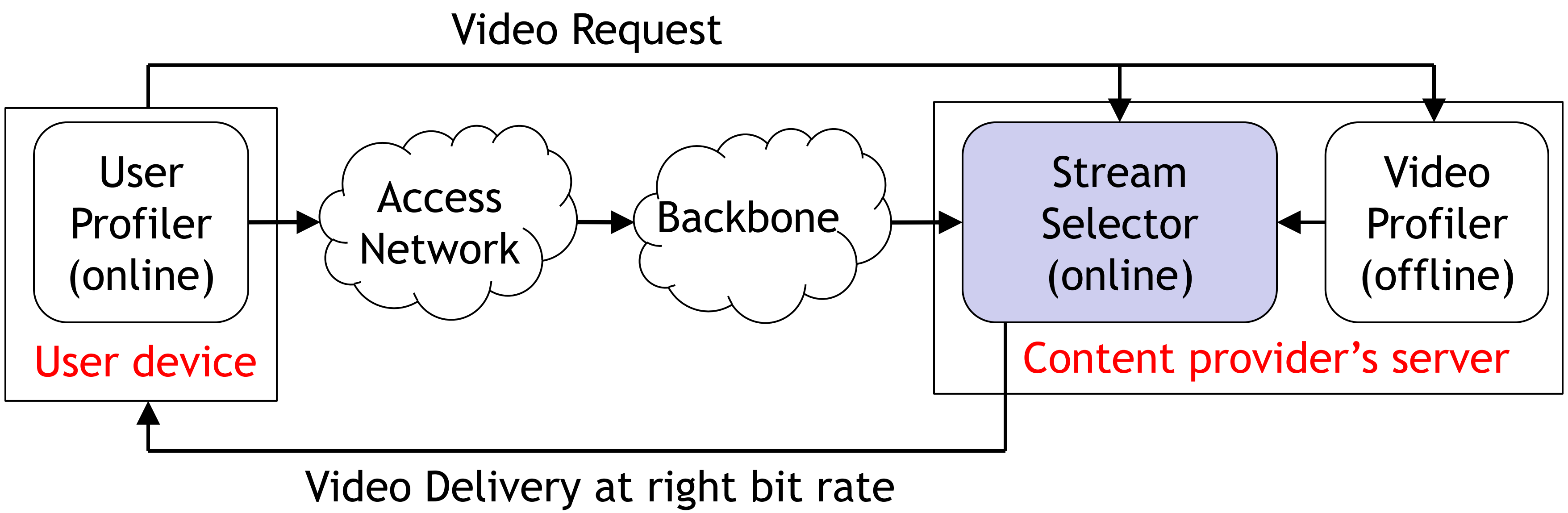
Usage profile

Modular Architecture

- Three Modules
  - Video Profiler
    - Exploit video compressibility from motion vectors
  - User Profiler
    - Predict user's future data consumption from past history
  - Stream Selector
    - Choose the right bitrate to maximize video quality subject to budget



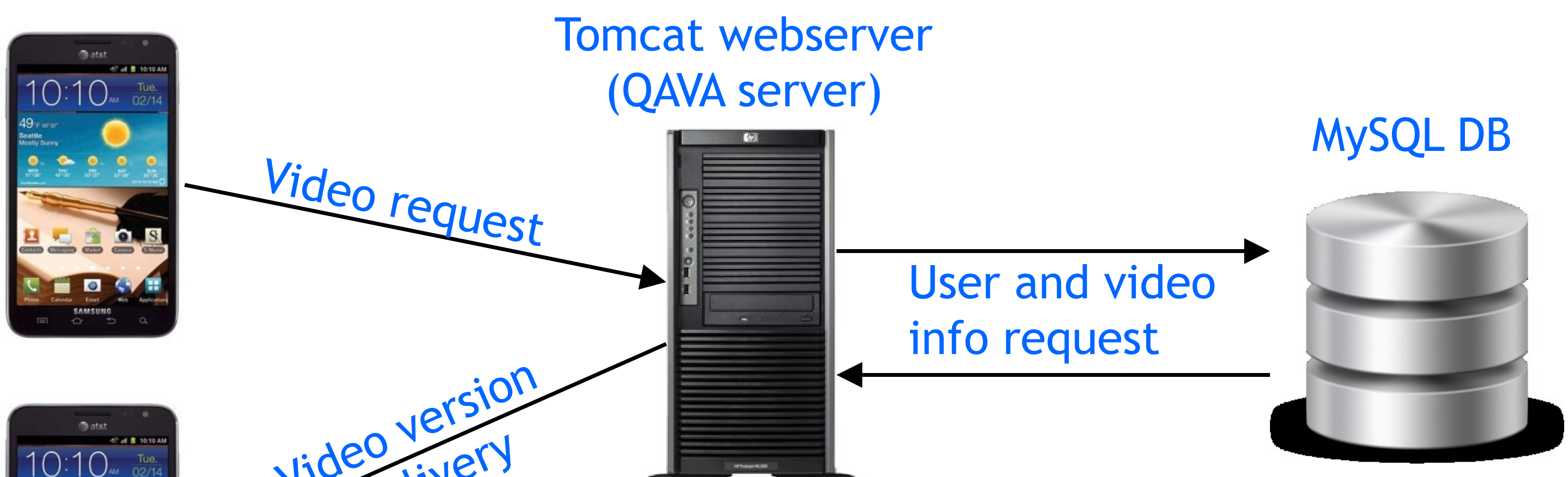
Modular Architecture



Adaptively choose the right bit rates

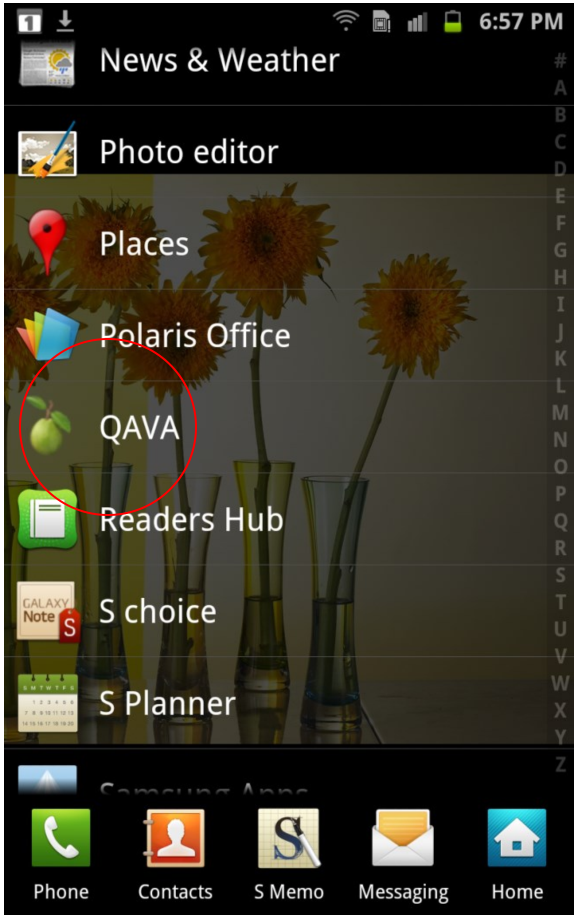
Princeton Trial

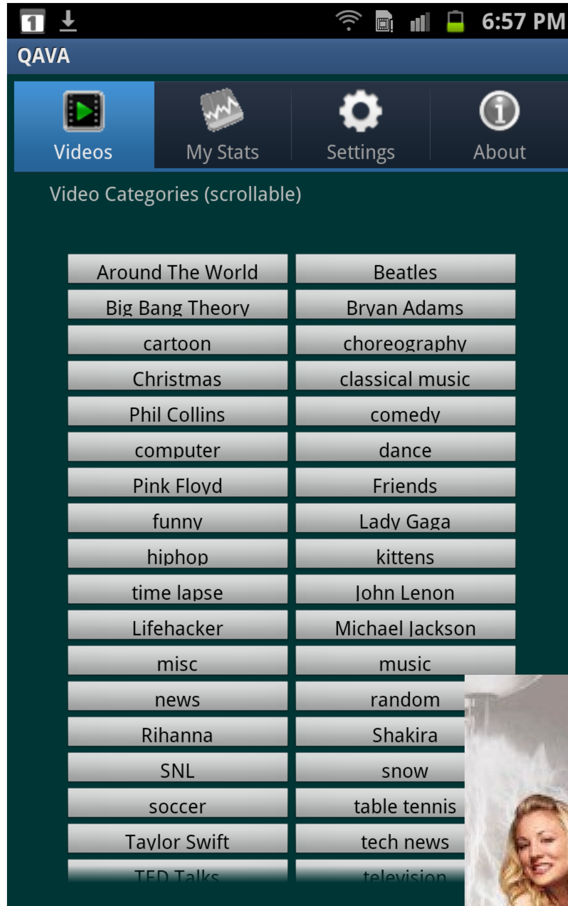
- Set Up
  - 15 volunteers with Android phones
  - ~500 videos encoded at 25 Kbps granularity (100 Kbps – 500 Kbps)

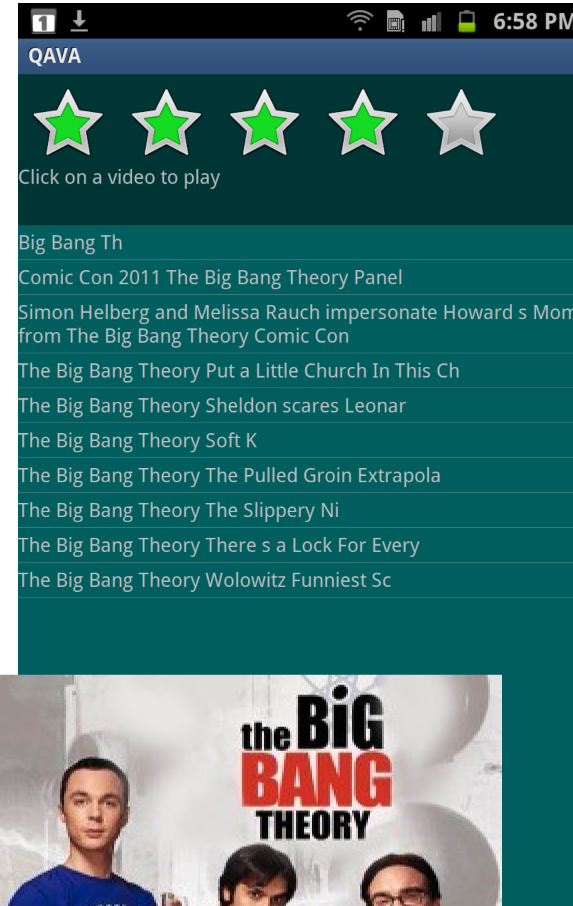


Database logs:
- Video request
- Time stamp
- User ID / Android ID
- MB of video delivered

Android App Screenshots



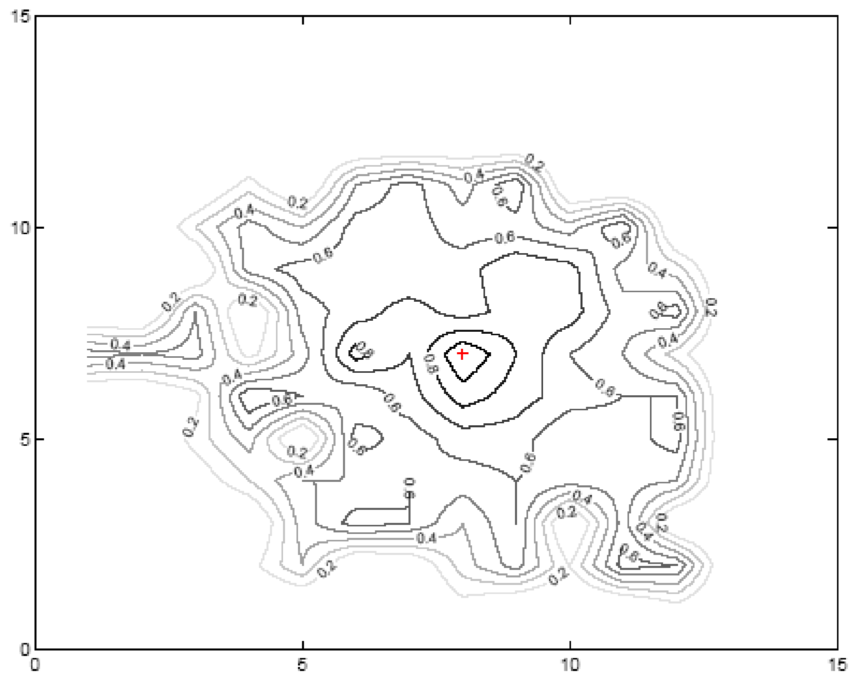




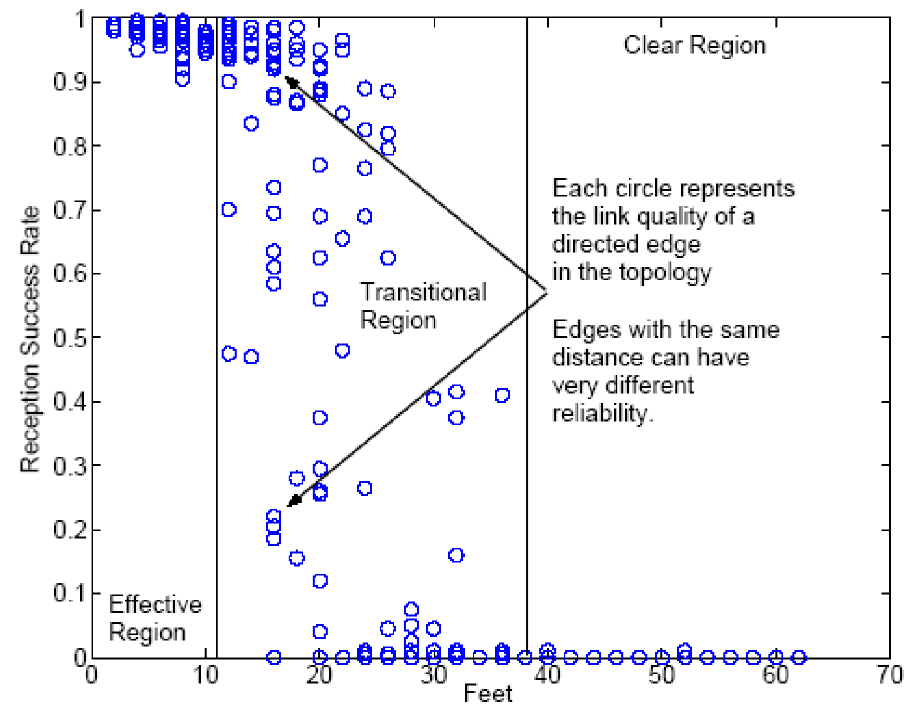
Outline

- Administrative Stuff
- Wireless Links
- GPS and Localization in Sensor Networks
- Open Forum

Reality

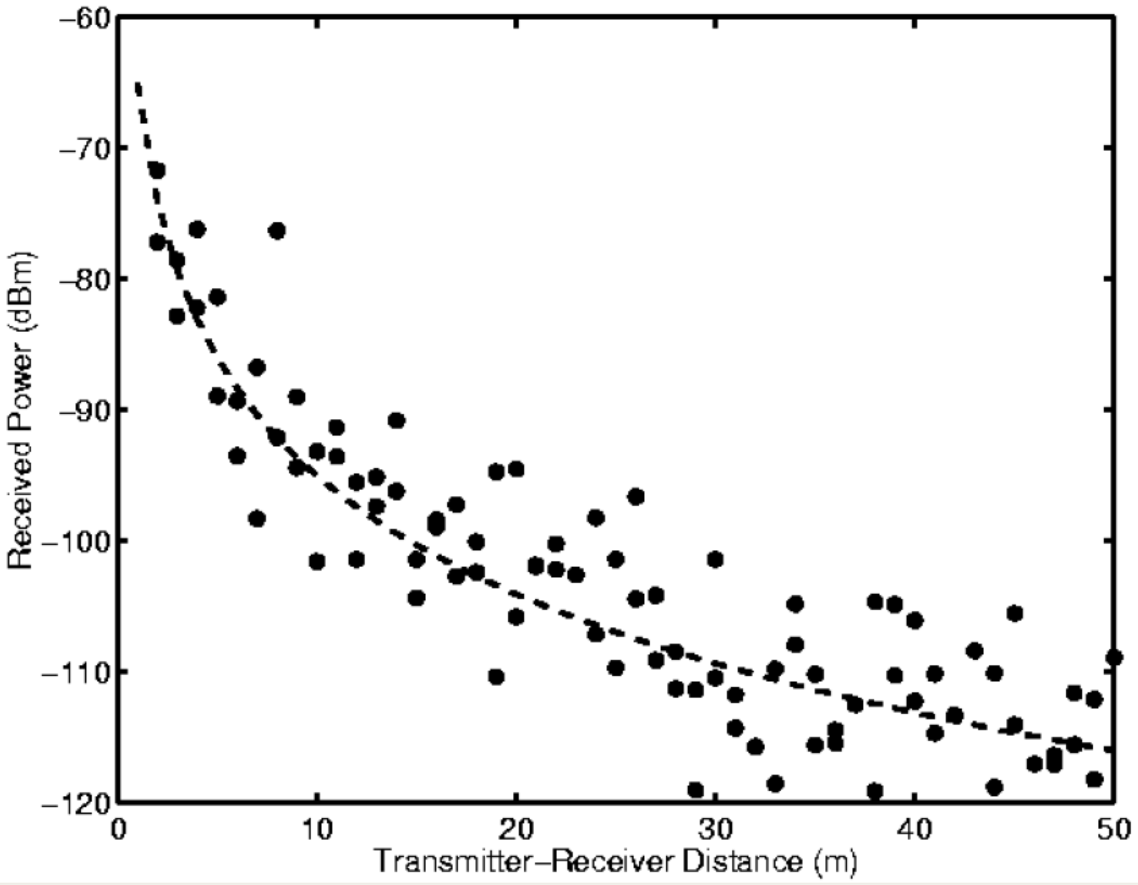


Ganesan et al. ‘02



Woo et al. ‘03

Radio Propagation



A Simple Model

- Exponential path loss with log-normal fading:

$$P_{r,dB}(d) = P_{t,d}B - PL_{dB}(d)$$

$$PL_{dB}(d) = PL_{dB}(d_0) + 10n \log_{10} (d/d_0) + X_{\sigma,dB}$$

Localization Overview

- Localization – To determine the location of objects

- Location information is necessary / useful for many functions
  - Location stamps
  - Coherent signal processing
  - Tracking and locating objects
  - Cluster formation
  - Efficient addressing
  - Efficient querying and routing

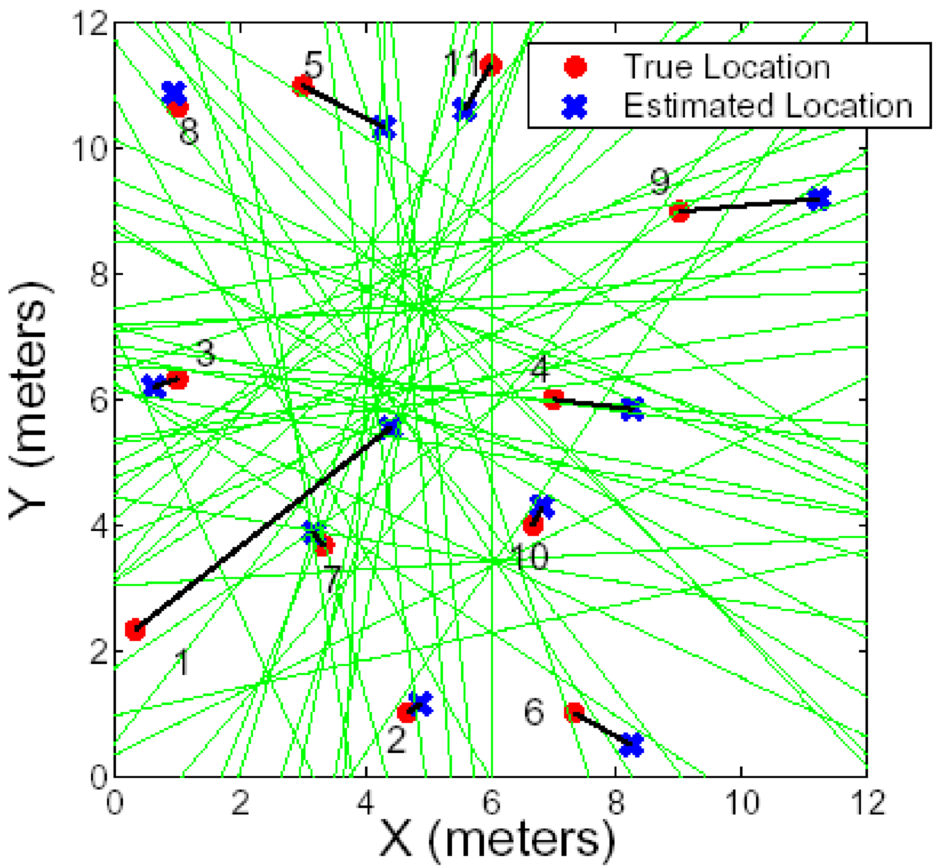
Localization Design Issues

- What to localize?
  - Unknown node vs. reference node
  - Mobile vs. static node
  - Node localization vs. network localization
  - Cooperative vs. non-cooperative nodes
- When to localize?
  - Static vs. dynamic
- How well to localize?
  - Coarse vs. fine grained
- Where to localize?
  - Central server vs. localizing object
- How to localize?
  - Technology: RF, IR, Ultrasound, Combination, UWB
  - What methodology to use?

Node Localization Approaches

- Coarse-grained
  - Use minimal information
  - Use minimal computation power
- Fine-grained
  - Gather and use as much information as possible
  - Requires higher computation power
- Trade-off
  - Accuracy vs. implementation / computation / cost

Radio Signal Strength Based Localization



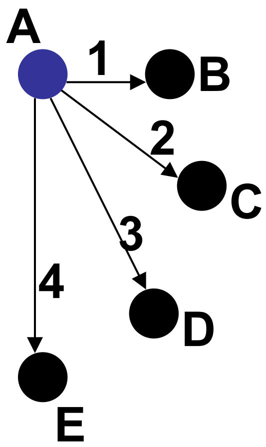
- Developing local positioning systems suitable for embedded wireless devices
- Low cost alternatives to GPS that can also work well under foliage / indoor environments
- Ecolocation and sequence-based localization

Ecolocation

- Unknown node initiates localization process
  - Sends out a localization request
- Reference nodes in the radio range send response packets
- Measure signal strength of received packets (RSSI)
- Rank reference nodes based on RSSI values
  - Ranks can be written as a set of constraints on the location of the unknown node
- The locations of reference nodes with respect to the grid points can also be written as distance constraints

Location Constraints

- Relationship between distances of a pair of reference nodes with respect to the unknown node
  - N reference nodes => n(n-1)/2 constraints (A constraint set)

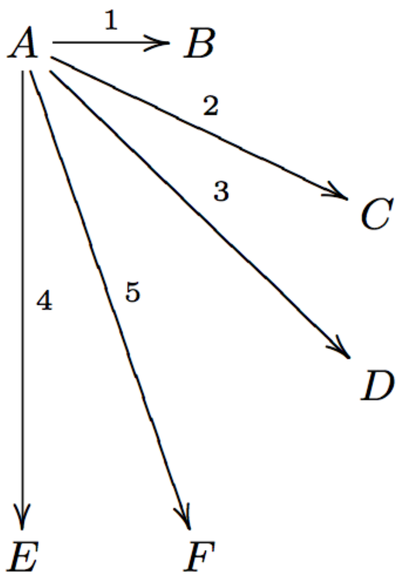


Location Constraint Set for A

$$\{d_B < d_C,\ d_B < d_D,\ d_B < d_E,\ d_C < d_D,\ d_C < d_E,\ d_D < d_E\}$$

Redundancy in the constraint set

Location Constraints



B:1	C:2	D:3	E:4	F:5
R_1	$R_2 < R_1$	$R_3 < R_1$ $R_3 < R_2$	$R_4 < R_1$ $R_4 < R_2$ $R_4 < R_3$	$R_5 < R_1$ $R_5 < R_2$ $R_5 < R_3$ $R_5 < R_4$

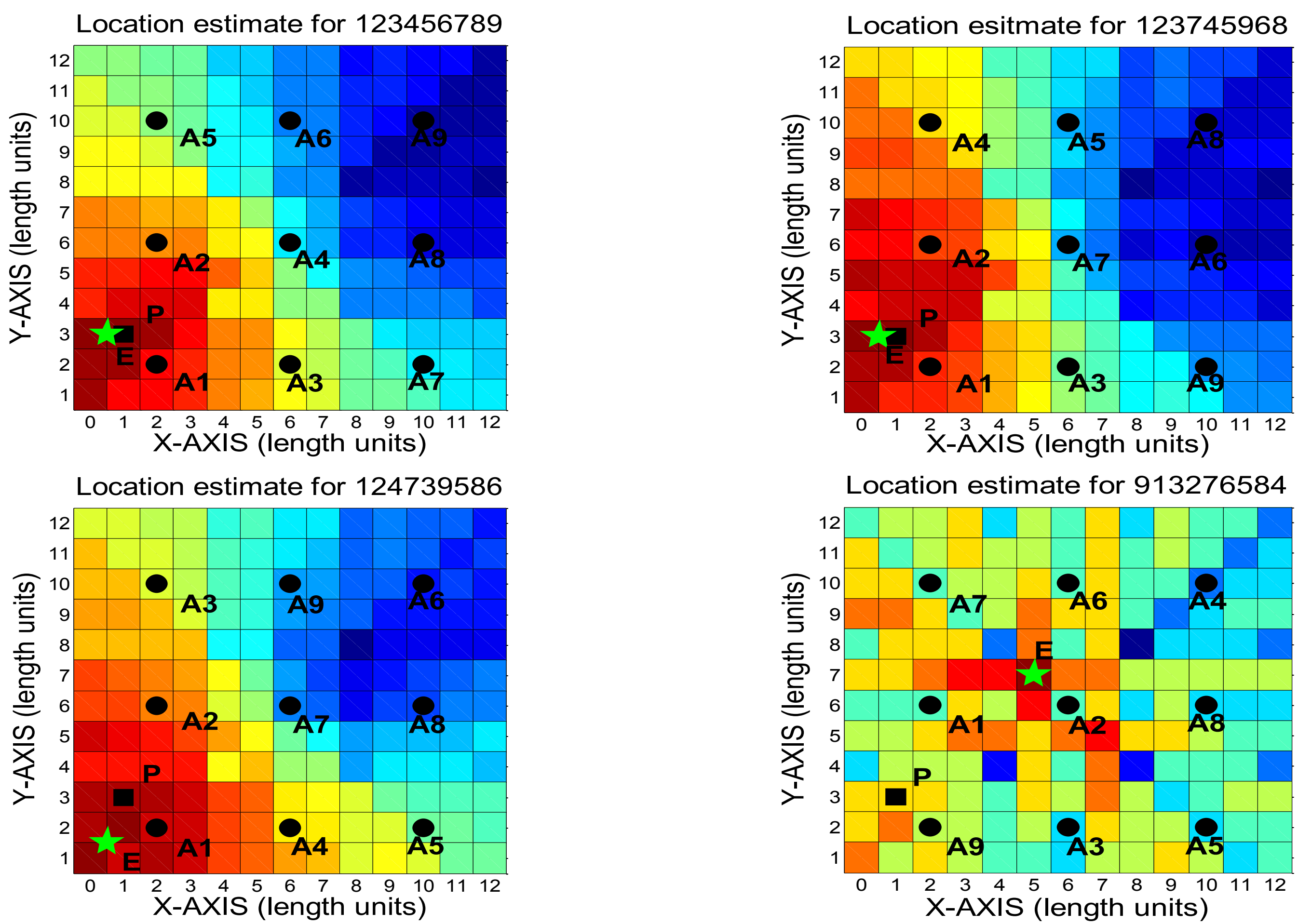
Constraints on the unknown node w.r.t. the reference nodes

$$M_{\alpha\times\alpha}(i,j) = \begin{matrix} 1 & \text{if } R_i < R_j \\ 0 & \text{if } R_i = R_j \\ -1 & \text{if } R_i > R_j \end{matrix}$$

Constraints on the reference nodes w.r.t. each of the grid points

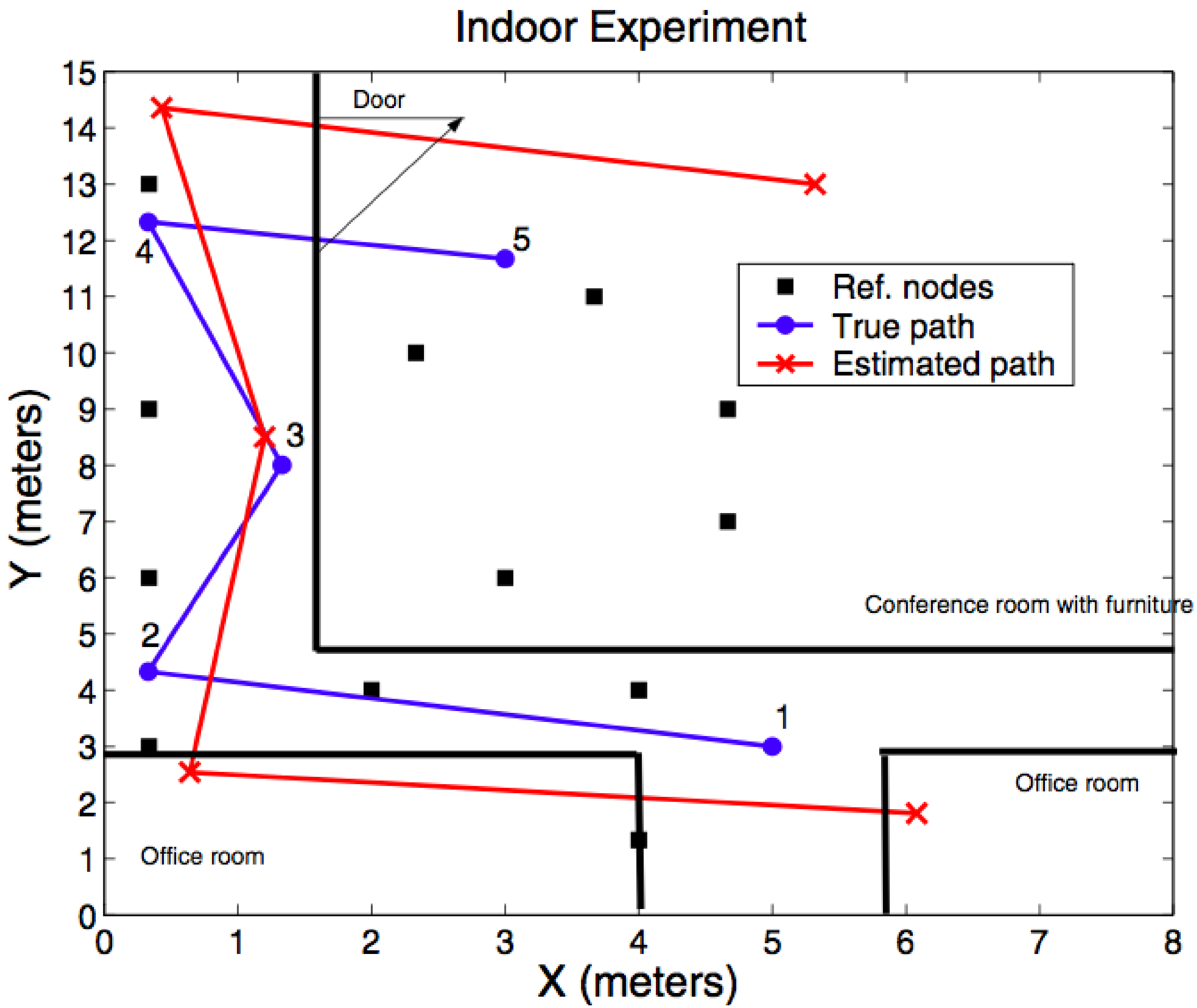
$$C^{ij}_{\alpha\times\alpha}$$

Ecolocation Results

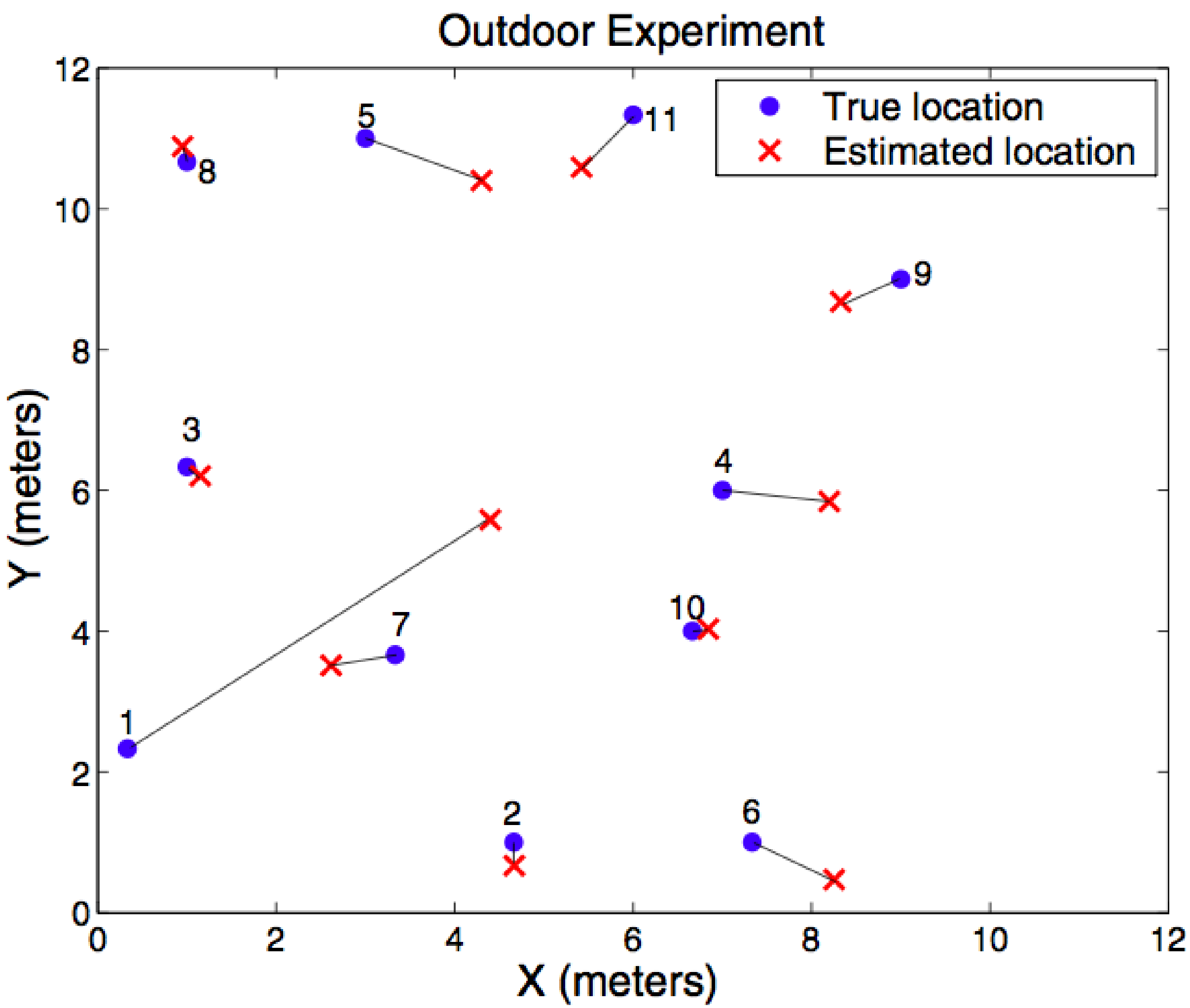


A: Reference Node P: True Location of unknown node E: Ecolocation Estimated Location

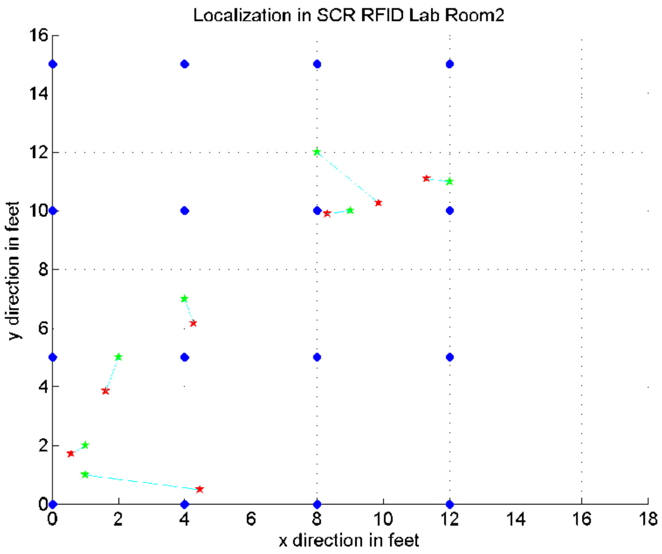
Indoor Tracking

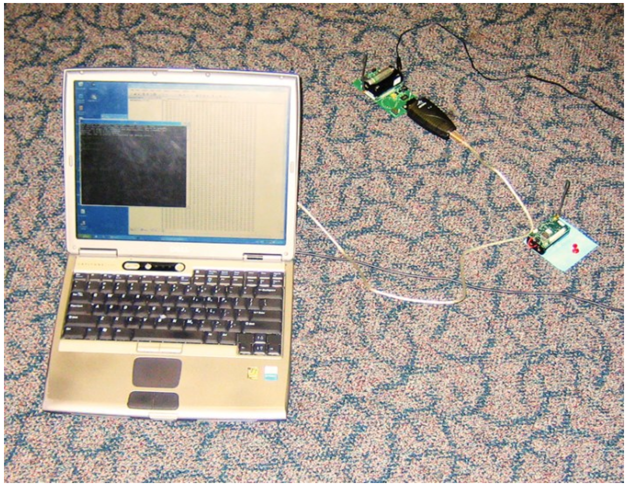


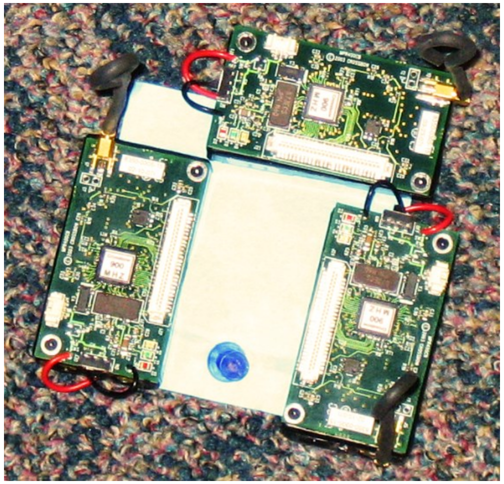
Outdoor Experiment



Maximum Ratio Combining

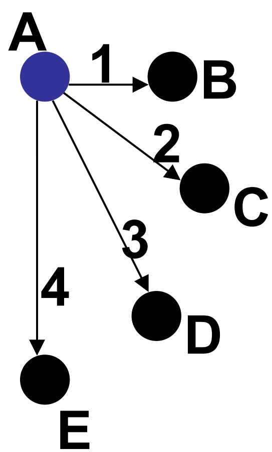






Location Sequence

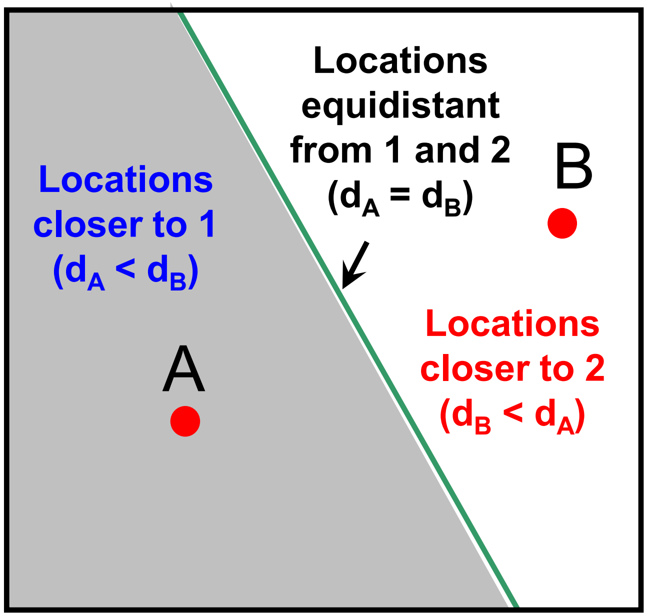
- The ordered sequence of distance ranks of reference nodes from a given location



Location Sequence for A

B C D E

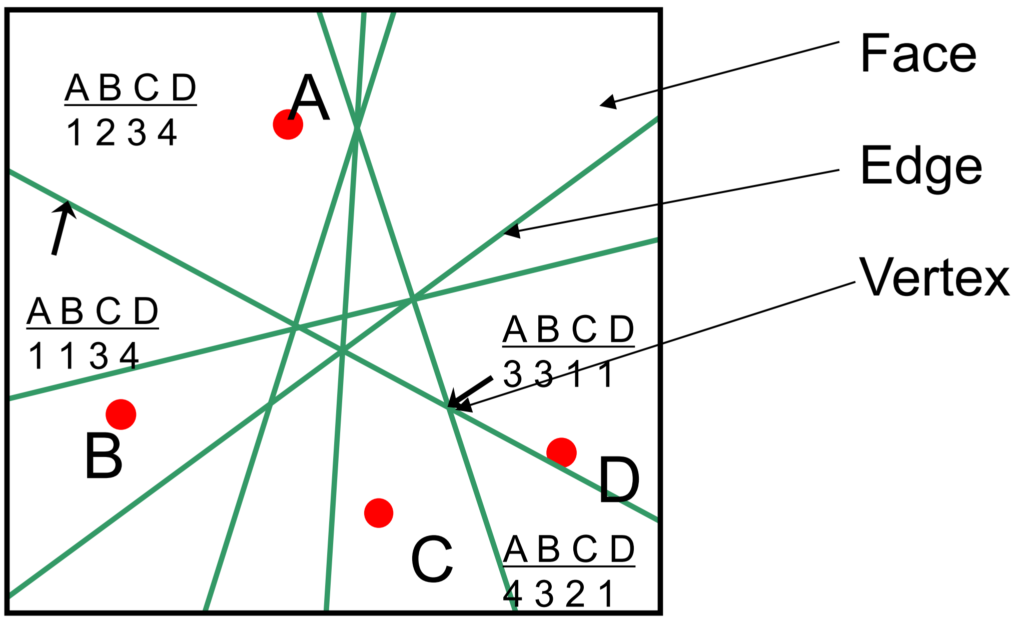
1 2 3 4



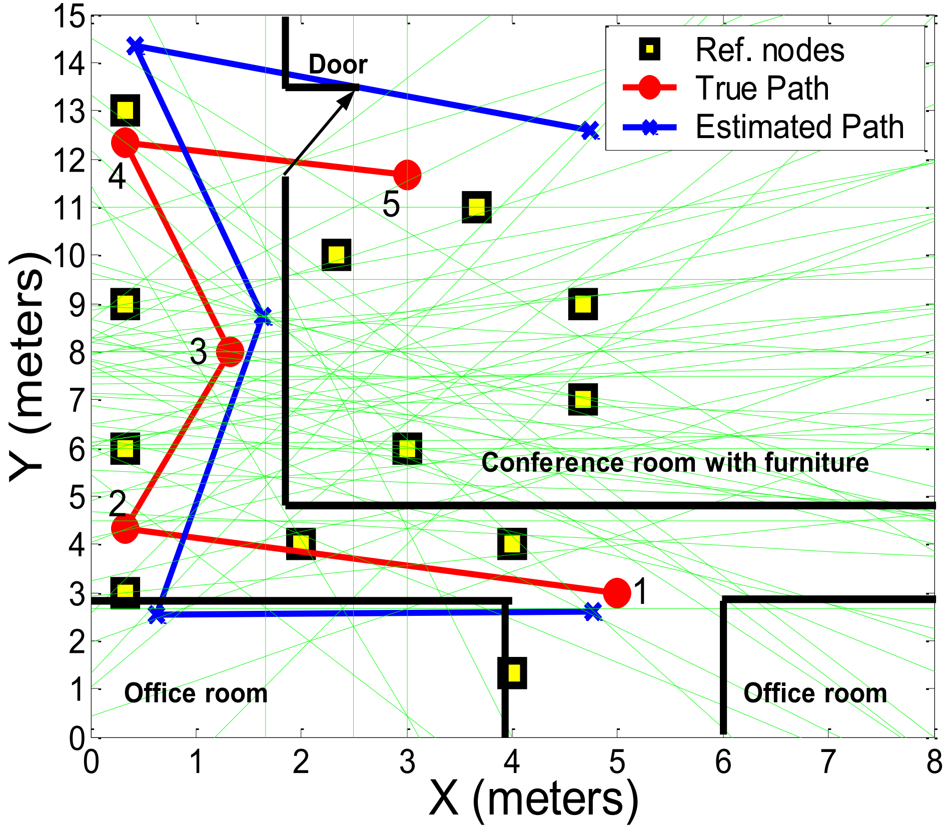
Rank order between two reference nodes is defined by the perpendicular bisector between them.

Location Sequence

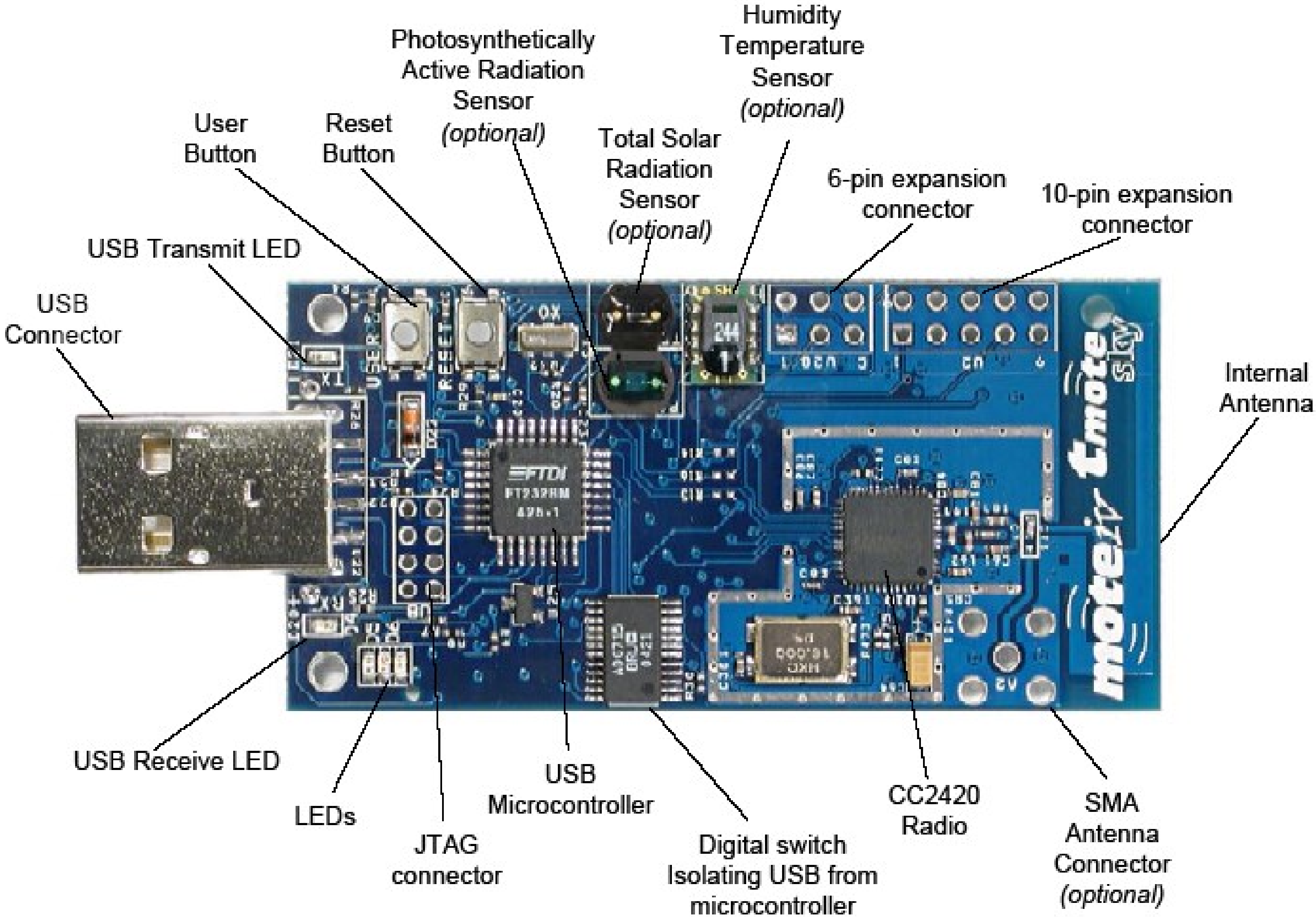
- Location sequences are unique to each region
- All locations in a region have the same location sequence
- One-to-one mapping with centroid of the region they represent



Indoor Experiment: Office Building



Moteiv Tmote Sky



Tmote Sky Features

- 2.4 GHz, 250 Kbps IEEE 802.15.4 CC2420 radio, range: tens of meters
- MSP430 microcontroller, 10 KB RAM, 48 KB flash
- 1 MB external flash
- USB programming using NesC/TinyOS/Contiki
- On-board Humidity, Temperature, and Light Sensors
- Power consumption @ 3V: mcu + radio: ~20 mA, mcu alone: ~2 mA, standby: 20 μA

Recap









Contiki OS

- Open source - BSD license
  - Multitasking using C programming language
- Developed by Adam Dunkels at the Swedish Institute of Computer Science
  - Version 1.0 released in March 2003
  - Version 2.7 released November 15, 2013
- Highly portable
  - Tmote Sky, JCreate, TelosB, Atmel Raven, MicaZ, …
  - Simulators: Cooja, MSPsim, AvoraZ, netsim
  - Native platform
- Actively developed
  - 17 developers from SICS, SAP, Cisco, NewAE, TU

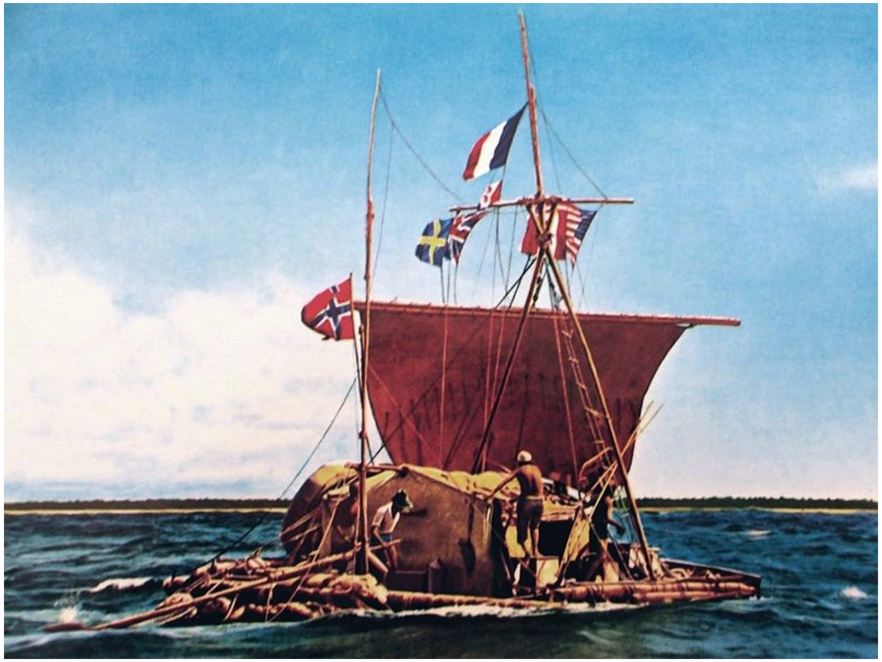
Contiki OS (2002)

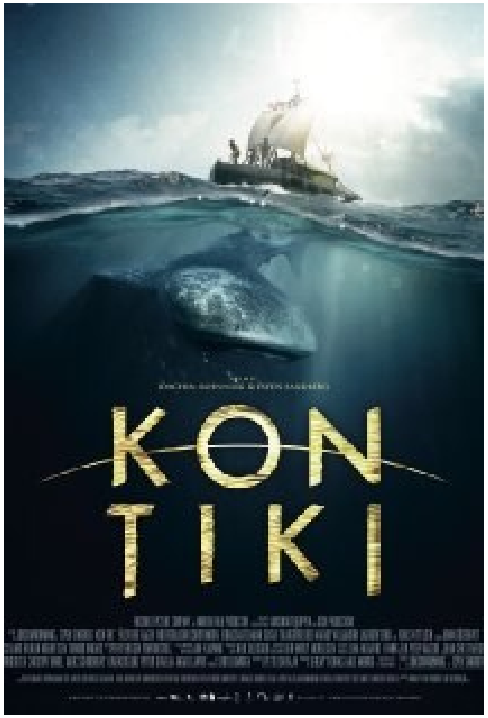
- Contiki – pioneering open source operating system for sensor networks
  - IP networking
  - Hybrid threading model, protothreads
  - Dynamic loading
  - Power profiling – measure network power consumption
  - Network shell – makes interaction easier
  - Rime stack – makes network programming easier
  - Multitasking using C language
  - Highly portable – 14 platforms, 5 CPUs

- Small memory footprint targeted for small embedded processors with networking
  - 50% of all processors are 8-bit, e.g., MSP430, AVR, ARM7, 6502, ...

The Name Contiki

- The Kon-Tiki raft
  - Used by Norwegian explorer and writer Thor Heyerdahl in his 1947 expedition across the Pacific Ocean from South America to the Polynesian islands with minimal resources
  - Named after the Inca sun god, Viracocha, whose old name was “Kon-Tiki”





Getting Started

- Step 1: Download Instant Contiki
  - Contiki development environment – single-file download
  - Ubuntu Linux virtual machine with all development tools, compilers, and simulators installed
  - www.contiki-os.org/start.html

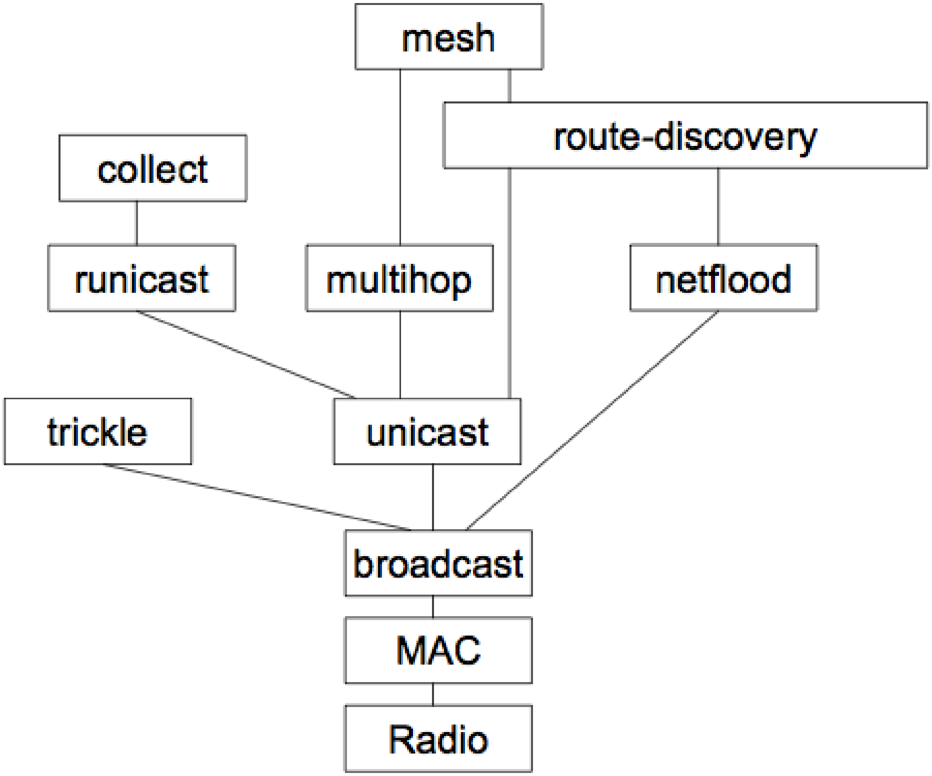
- Step 2: Download VMWarePlayer
  - www.vmware.com/go/downloadplayer

- Step3: Start Instant Contiki
  - Open the Instant Contiki folder and execute
  - instantContiki2.6.vmx
  - Wait for the virtual Ubuntu Linux to boot up

Rime: Lightweight and Layered

- Each module is fairly simple
  - Compiled code 114-598 bytes
- Complexity handled through layering
  - Modules are implemented in terms of each other
- Not a fully modular framework
  - Full modularity typically gets very complex
  - Rime uses strict layering

http://contiki.sourceforge.net/docs/2.6/examples.html



An Example

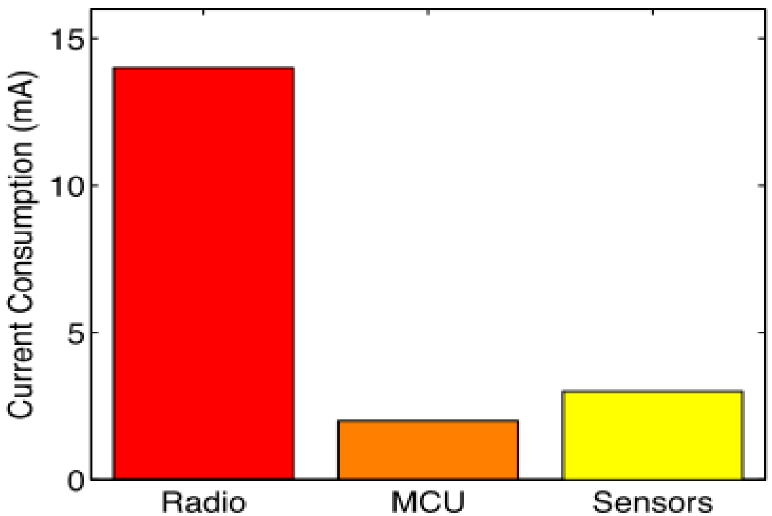
- We will go through an example Contiki program step-by-step to see the structure of the code and different data structures used

- This example program opens a UDP broadcast connection and sends one packet every second

Power Consumption

One of the biggest challenges

- Sensors have a limited source of power and it's hard to replace or recharge, e.g., sensors deployed in the battle field, sensors in a large forest



Radio mode	Power consumption (mW)
Transmit (T_x)	14.88
Receive (R_x)	12.50
Idle	12.36
Sleep	0.016

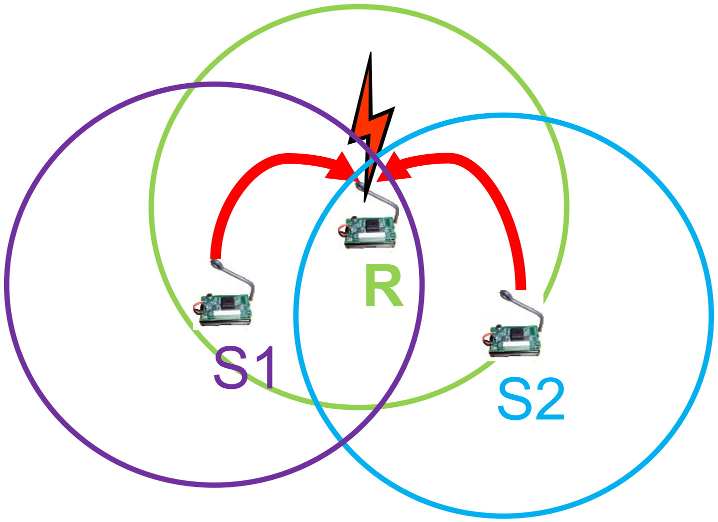
Sources of Power Consumption

Wasteful power consumption

- Idle listening to the channel
  - Waiting for possible traffic
- Retransmitting because of collision
  - Two packets arrived at the same time at the same sensor
- Overhearing
  - When a sensor received a packet doesn’t belong it
- Generating and handling control packets.

Hidden Terminal Problem

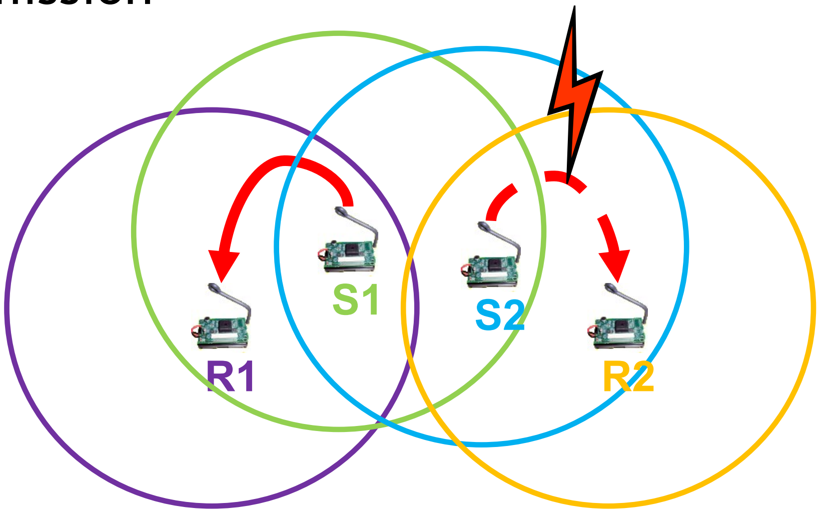
- Another sender's presence is hidden from the intended sender, and therefore simultaneous transmissions from both os them to the same receiver cause collision



- How to avoid? – Use of additional signaling packets
  - Sender asks receiver whether it is able to receive a transmission - Request to Send (RTS)
  - Receiver agrees, sends out a Clear to Send (CTS)
  - Sender sends, receiver sends Acknowledgements (ACKs)

Exposed Terminal Problem

- An exposed node is one that is in the range of the transceiver but not the receiver
  - Sender mistakenly thinks that the medium is in use, and it unnecessarily defers transmission

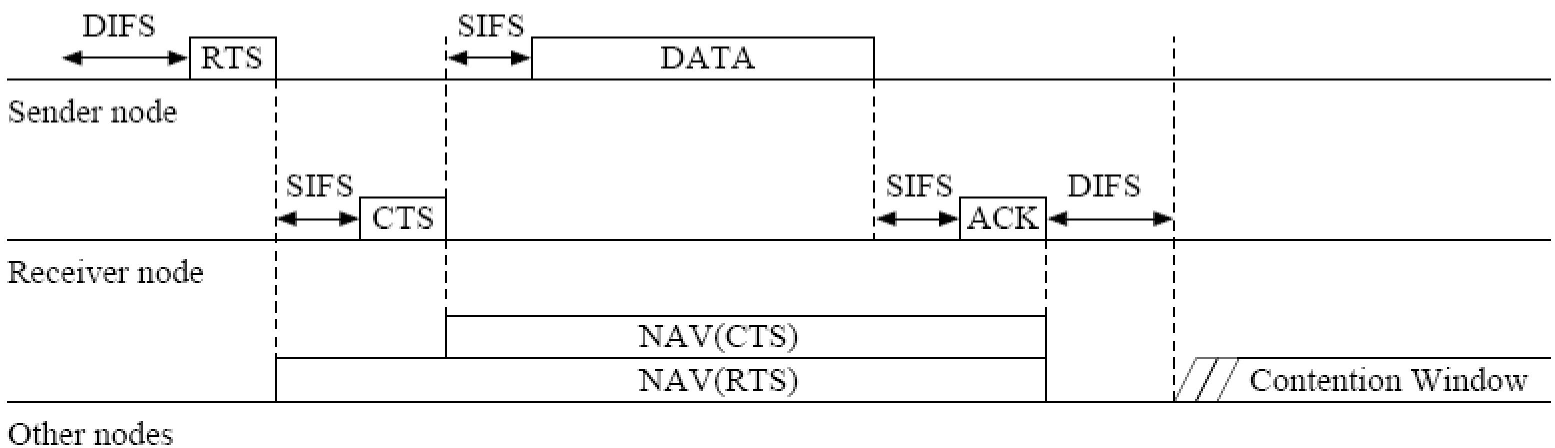


- How to avoid?
  - When a node hears an RTS but not a corresponding CTS, it can deduce that it is an exposed terminal and is permitted to transmit
  - Directional antennas

Wireless MAC Protocols

IEEE 802.11 Disadvantages

- Devices consume large amounts of energy due to the high percentage of time spent listening without receiving messages

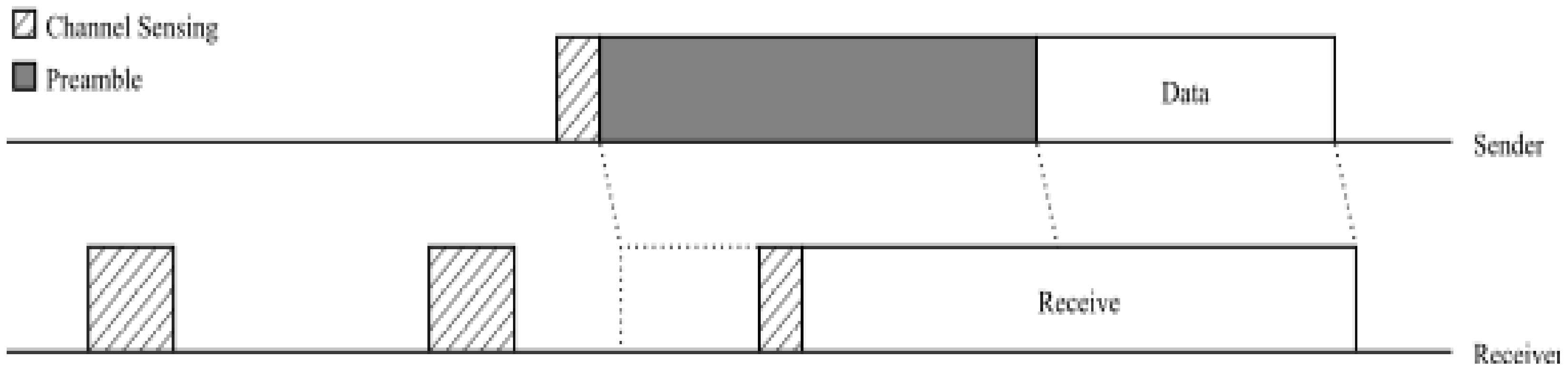


802.11 Data Transfer

Unscheduled WSN MAC Protocols

B-MAC

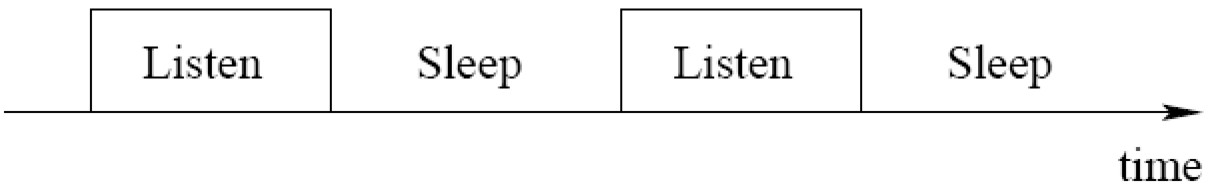
- ❑ Uses a tone to wake up sleeping neighbors, similar to STEM-T
- ❑ Uses very long preambles - dominates energy usage
- ❑ Suffers from overhearing problem



Scheduled WSN MAC Protocols

S-MAC (Sensor-MAC)

- Inspired by PAMAS, but in-channel signaling
- Nodes periodically go to a fixed listen/sleep cycle



- Virtual clustering to synchronize nodes on a common slot
- Energy is still wasted during listen period, as the sensor remains awake even if there is no reception/transmission

Scheduled WSN MAC Protocols

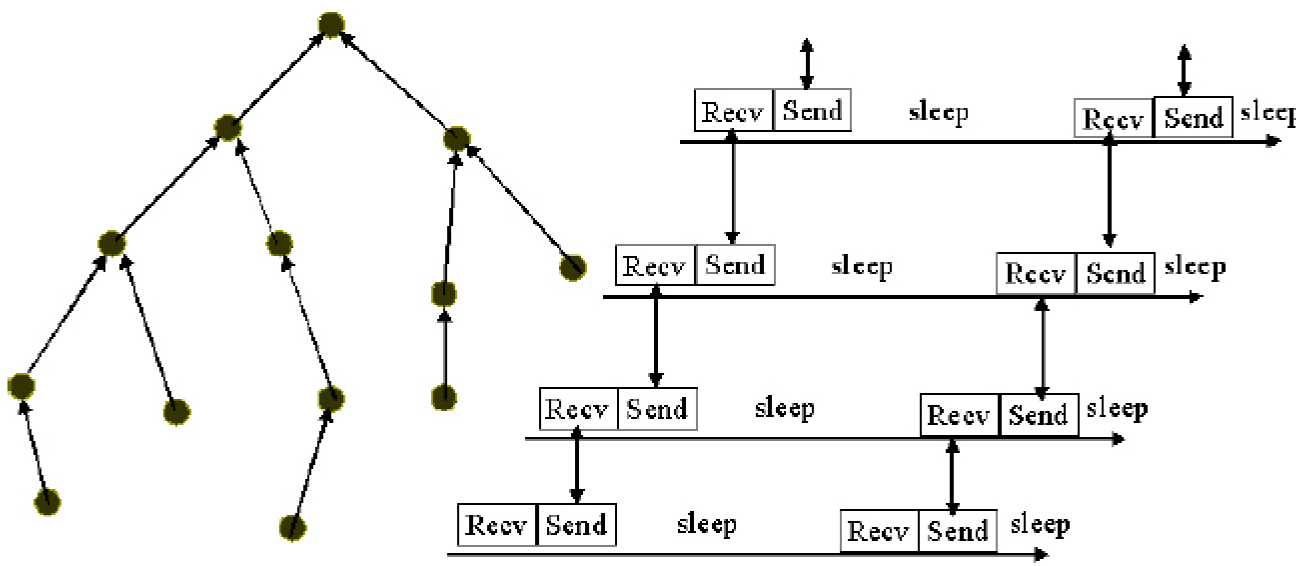
T-MAC (Timeout-MAC)

- Introduces adaptive duty cycling to improve S-MAC
  - Frees the application from the burden of selecting an appropriate duty cycle
  - Automatically adapts to traffic fluctuations
- Borrows virtual clustering from S-MAC for synchronization
  - Operates on a fixed length slot (615 ms)
  - Uses a time-out mechanism to dynamically determine the end of the active period
- Downside
  - Aggressive power-down policy (nodes often go to sleep too early)

Scheduled WSN MAC Protocols

D-MAC (Data Gathering-MAC)

- Uses adaptive duty cycling like T-MAC
  - 1 receive, 1 send, and n sleep slots
- Low node-to-sink latency: convergecast
- Divides time into short slots (10 ms) and runs CSMA/CA within each slot



Convergecast tree with matching, staggered DMAC slots

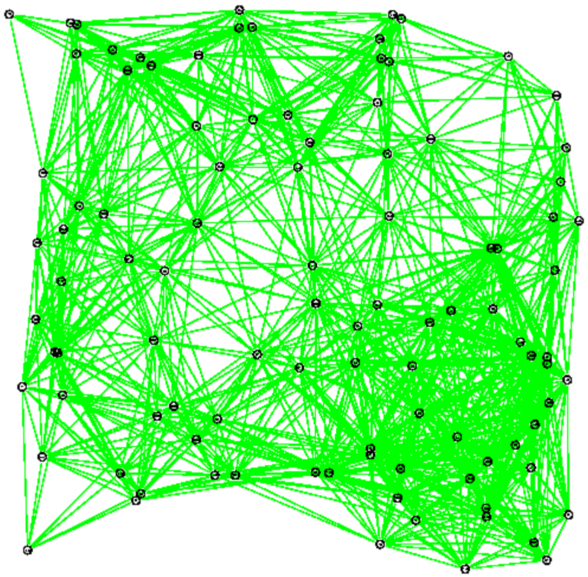
In Summary

There is no unique “best” MAC protocol for WSN. Each one is customized for specific applications.

Why Topology Control?

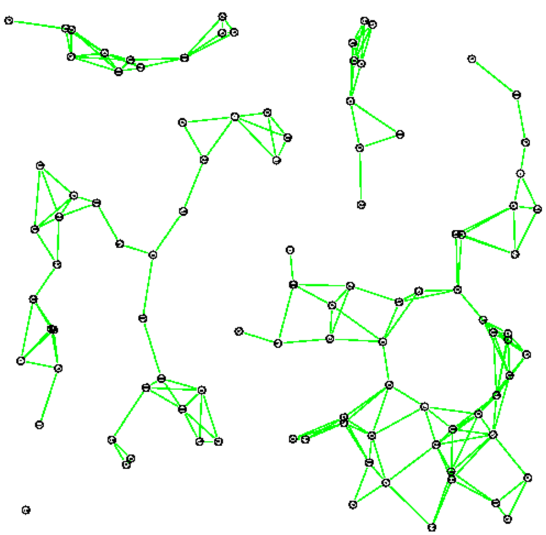
Topology Control: Given a network connectivity graph, compute a subgraph with certain properties: connectivity, low interference etc.

- No topology control: nodes transmit at max power levels



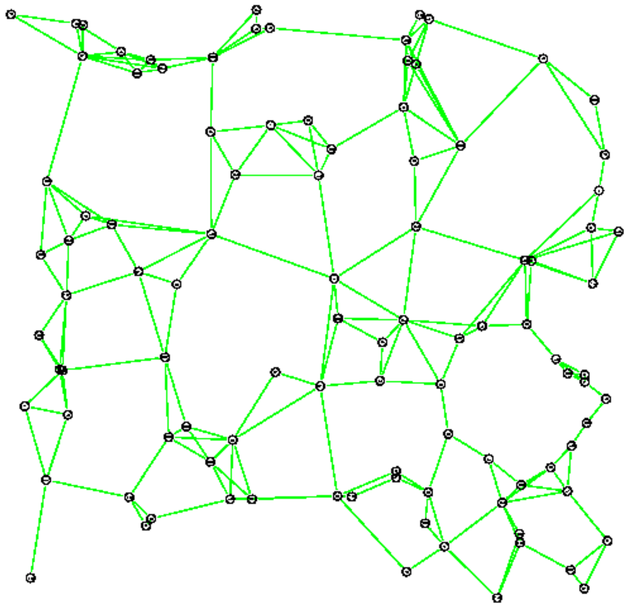
- High energy consumption
- High interference
- Low throughput

- No topology control: nodes transmit at min power levels



- Network may partition

An Example



Benefits

- Global connectivity
- Low energy consumption
- Low interference
- High throughput

Problem

- To find optimal transmission power levels using local information such that network connectivity is maintained.

Cone-Based Topology Control

2D CBTC

Global connectivity from local geometric constraints [Wattenhofer, Infocom ’01] [Li Li, PODC ’01, TON ’05]

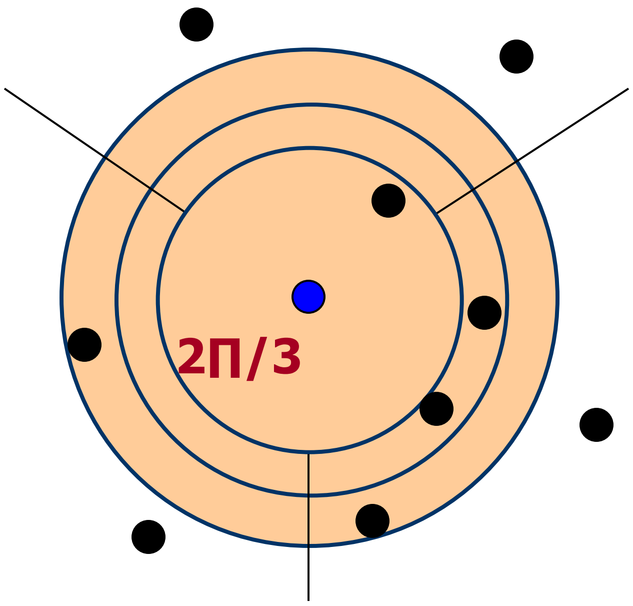
Assumptions

- Maximum Power Graph G=(V, E) is connected
- Assume receivers can determine direction of senders

Main Result

If every node adjusts its power level, such that there exists at least one neighbor at every **2Π/3** sector around itself, then network is connected

- Complexity **O(d log d),** d = avg node deg
- Not (efficiently) extensible to 3D



3D Topology Control

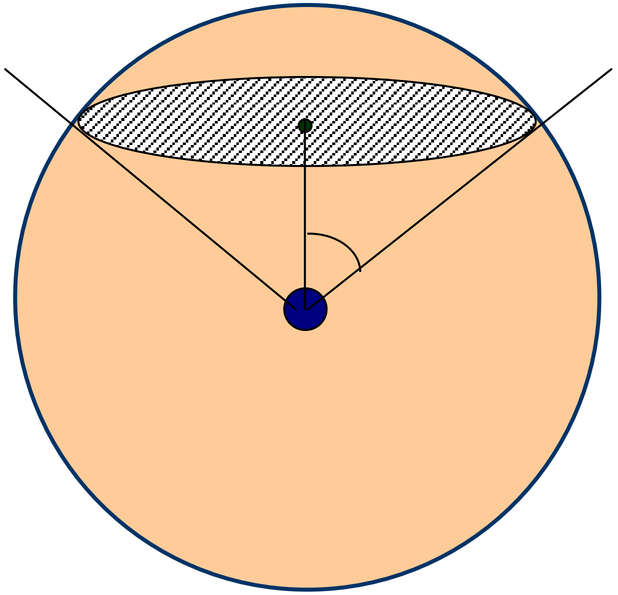
3D CBTC [Bahramgiri, ICCCN'05, Wireless Networks '06]

Basic Idea

Each node increases its power level until there is at least one neighbor at every **3D cone** of apex angle **$2\pi/3$** around it

Limitations

- Assumes directional information
- High time complexity – **$O(d^3 \log d)$**



Our Approach

- Phase 1
  - Use Multi-Dimensional Scaling (MDS) to find relative location maps for each node's neighbors when they use P^{max}
- Phase 2
  - **Simplify the 3D problem**
    - Orthographic Projections
      - Convert the 3D problem into similar problems in 2D
    - Solve the 2D problems using CBTC and infer about the 3D solution
  - **Solve the 3D problem directly**
    - Use Spherical Delaunay Triangulation (computational geometry tool)

Phase 2: Orthographic Projections

Algorithm:

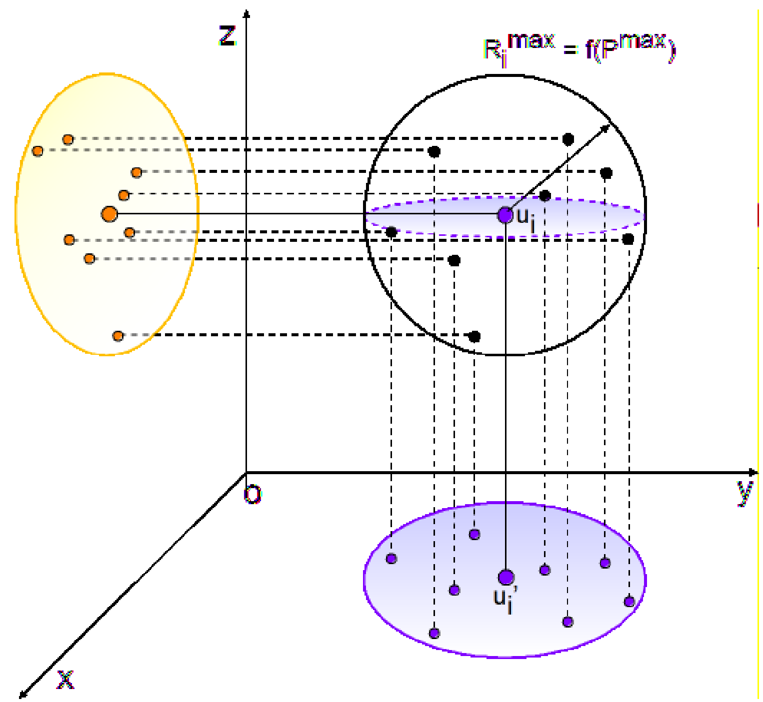
Hope that by satisfying CBTC on 3 planes => non-empty 3D cones

1. Each node starts with minimum tx. power

2. For a given tx power, project the neighbors on xy, yz, and zx

3. Run 2D CBTC on each plane
   - If any of the 3 planes do not satisfy the **2Π/3** constraint, increase power to the next level
   - Else STOP, settle with current power

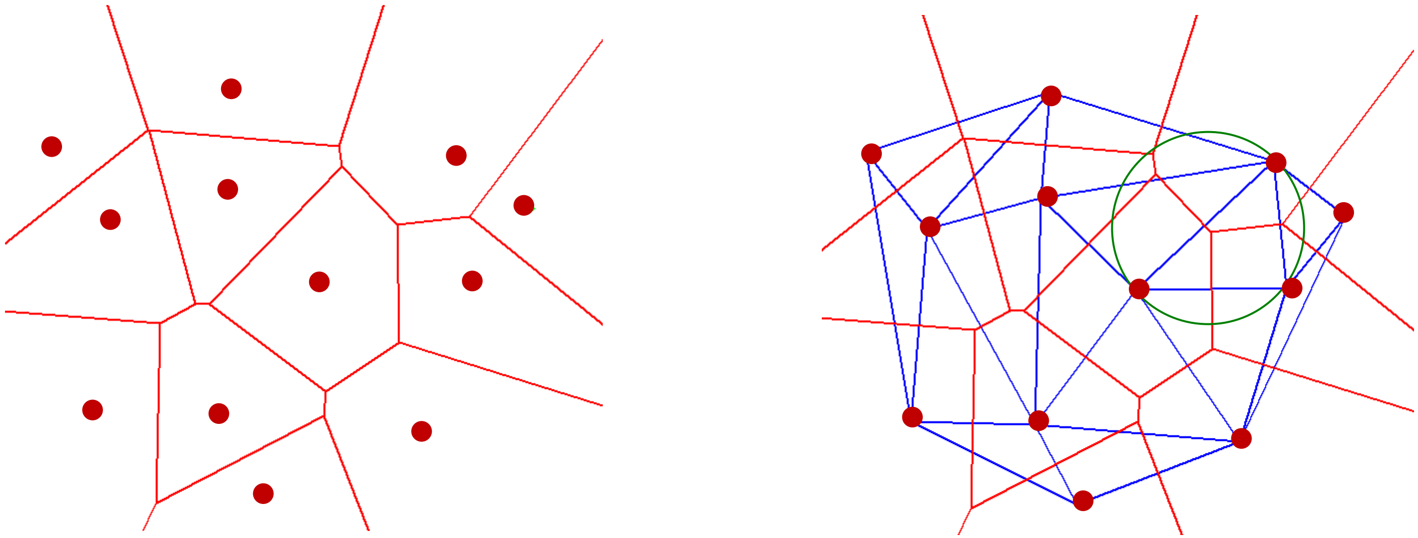
4. Go back to Step 2 unless P^{max} is reached.



Delaunay Triangulation

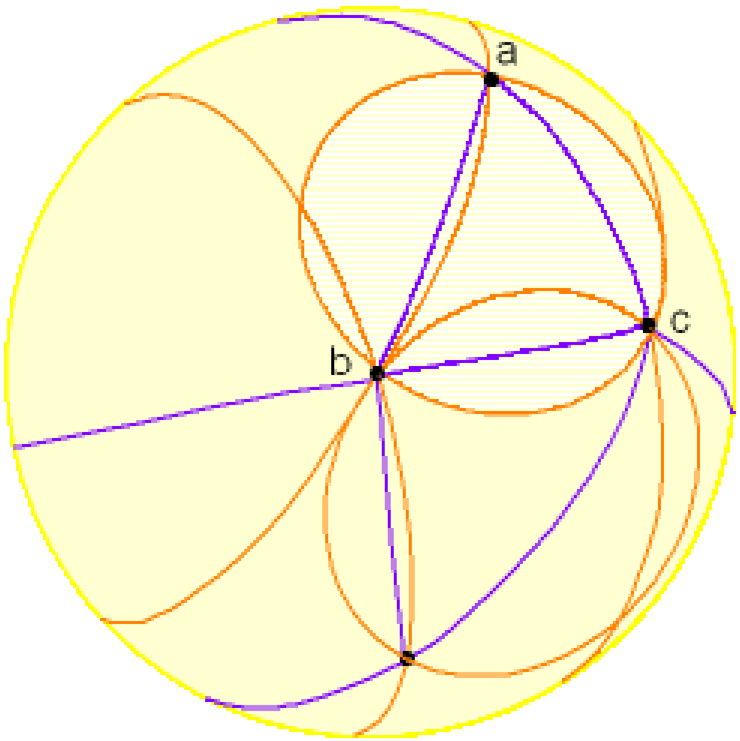
Dual of Voronoi diagram

Empty circumcircle property of DT



Spherical Delaunay Triangulation

- When we do the DT on the surface of a sphere

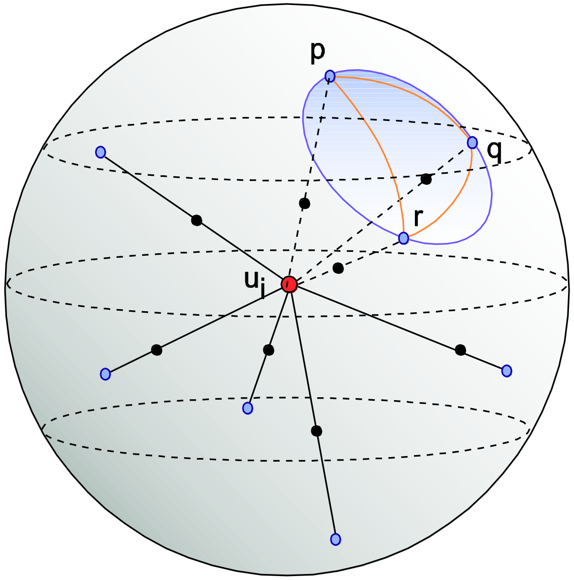


Spherical triangles,
and spherical caps

Phase II: SDT

Algorithm: SDT

1. Each node starts with minimum tx. Power
2. For a given tx. power, project the neighbors on the spherical surface
3. Construct Delaunay triangulation on the surface of the sphere
4. Calculate the area of the (empty) spherical caps
5. If any cap area is > 2.7 R^2
   - Increase the power to next level; go to Step 2
6. Else
   - Stop, settle down with current power level

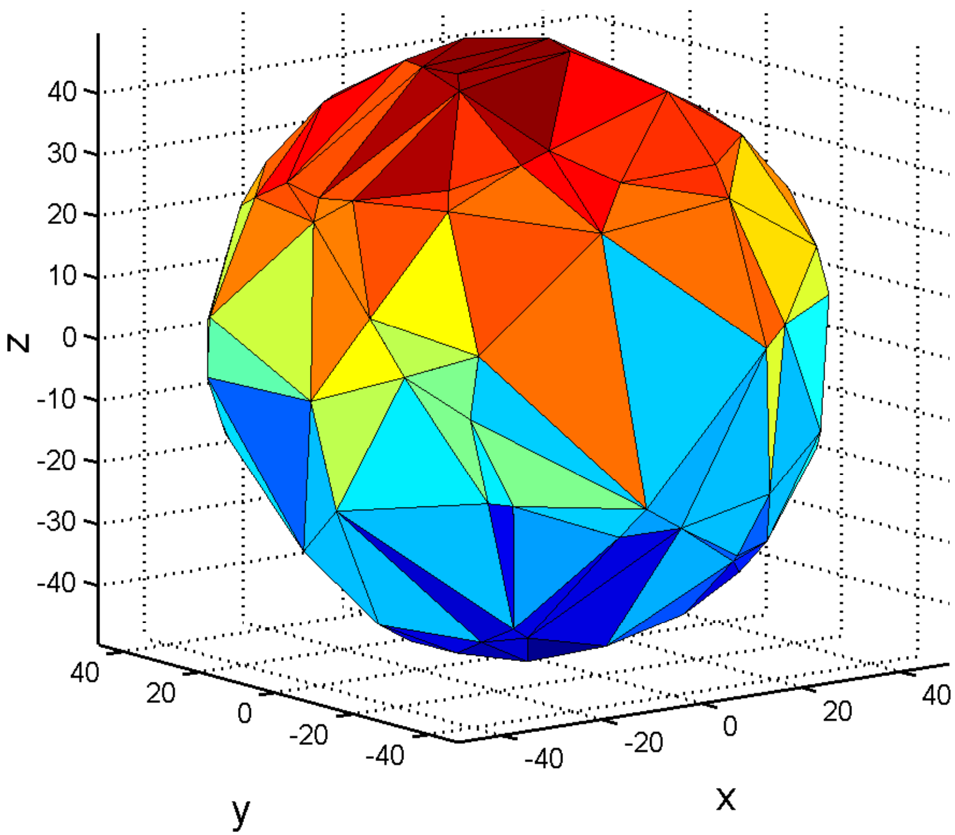


O(d log d)

Lemma 2

If none of the spherical caps have a surface area greater than **$2.7R^2$**, the network is at least one-connected.

Visualization of SDT in Matlab



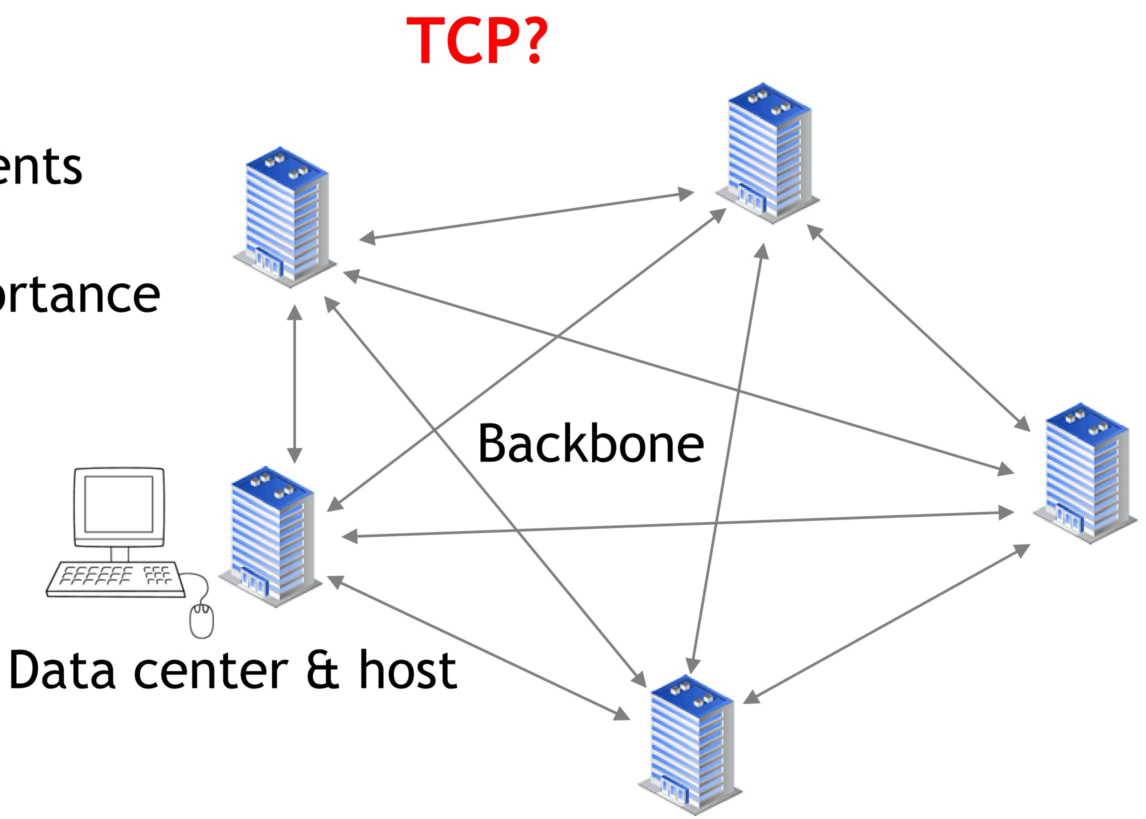
Spherical Delaunay Triangulation using Quickhull for 100 points randomly distributed on the surface of a sphere of radius 50

Scalable Multi-Class Traffic Management in Data Center Backbone Networks

(Collaborators: Google, Princeton)

Motivations

- ❑ Multiple interconnected data centers (DCs) with multiple paths between them
- ❑ DCs, traffic sources, and backbone owned by the same OSP, e.g., Google, Yahoo, Microsoft
- ❑ Traffic with different performance requirements
- ❑ Different business importance

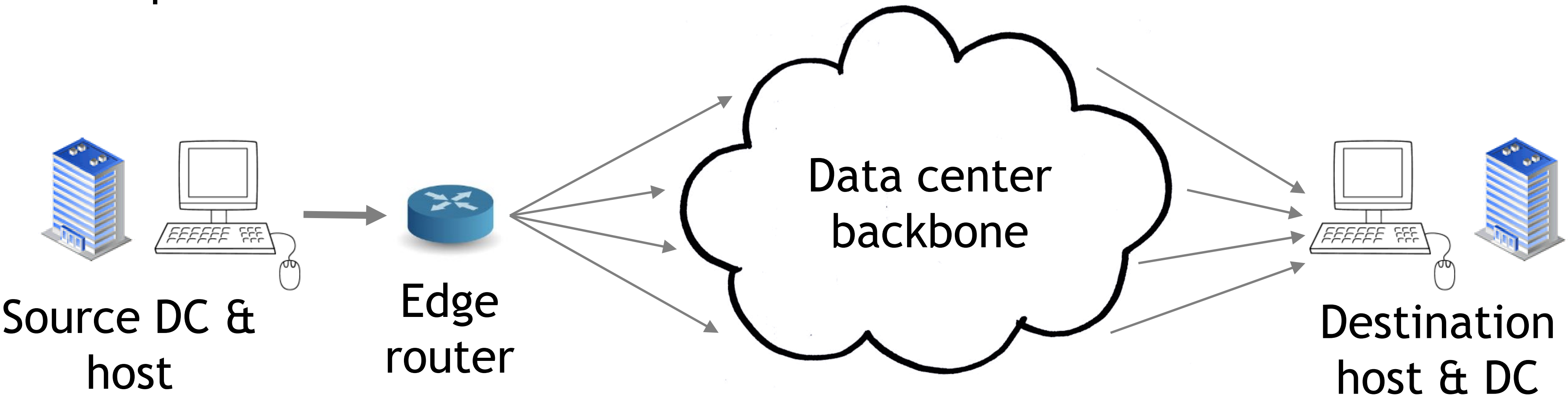


Contributions

Controlling the three "knobs"

- ❑ Sending rates of hosts
- ❑ Weights on link schedulers
- ❑ Splitting of traffic across paths

Joint optimization of rate control, routing, and link scheduling



Contributions

TRUMP (CoNEXT ‘07)
DaVinci (CoNEXT ‘08)

Fully-centralized
Not scalable

Semi-centralized: Our work
Modular, scalable with low message-passing and fast convergence

Fully-distributed
Scaling issues due to message passing, slow convergence

- Computation is distributed across multiple tiers using a few controllers
- Result is provably optimal using optimization decomposition
- Semi-centralized solutions viable and, in fact, preferred in practice, e.g., Google’s B4 globally-deployed software defined private WAN (SIGCOMM ‘13)

Model and Formulation

Utility of Flow s of Class k

Coefficients to model different degrees of sensitivity to throughput and delay

$$U_s^k = w_s^k \left[a^k f^k(x_s^k) - b^k g^k(u_l^k) \right]$$

Weight of flow s of class k

Throughput sensitivity of class k, e.g., log(.)

Total sending rate of flow s of class k

Delay sensitivity of class k

Utilization of class k over link l

Sum of the products of path rates and average end-to-end delays on those paths

Model and Formulation

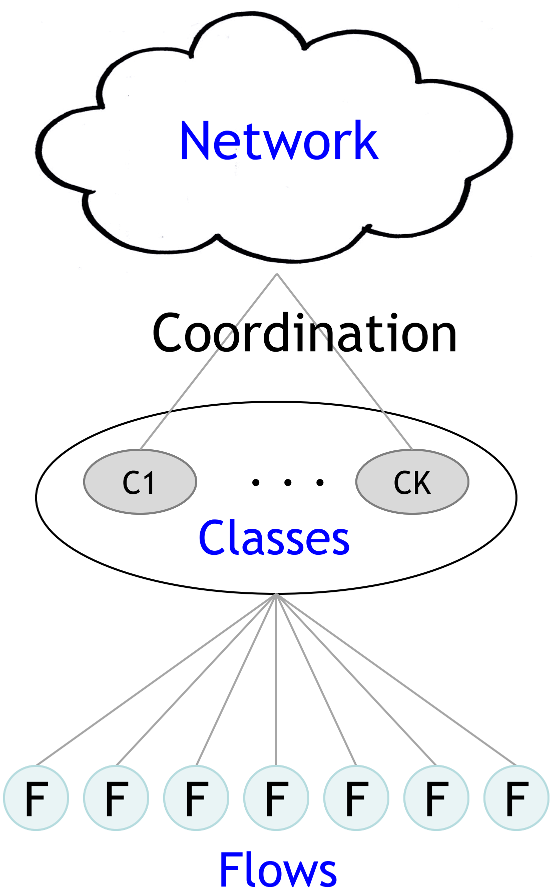
Objective Function

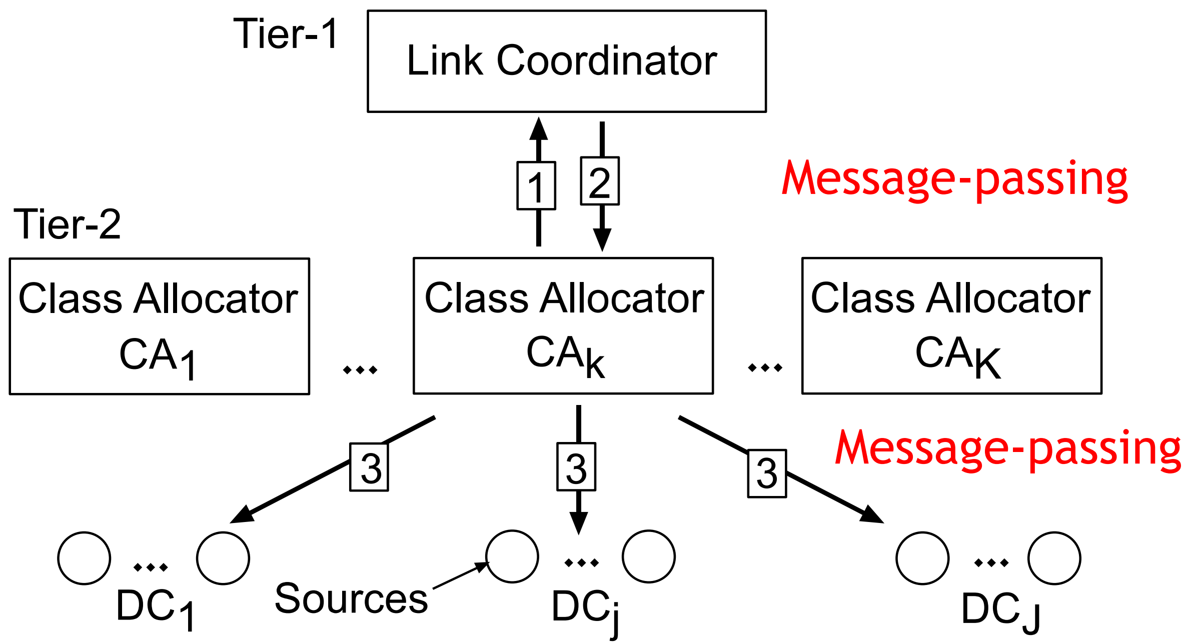
- Data centers, backbone and traffic sources under the same OSP ownership
- Maximize the sum of utilities of all flows across all traffic classes (global “social welfare”)

Global Problem G:

$$
\begin{aligned}
\text{maximize} \quad & \mathcal{U} = \sum_{k} \sum_{s \in \mathcal{F}^k} U_s^k \\
\text{subject to} \quad & \mathbf{A}\mathbf{R}^k \mathbf{z}^k \preceq \mathbf{y}^k, \quad \forall k \\
& \sum_{k} y_l^k \leq c_l, \quad \forall l \\
\text{variables} \quad & \mathbf{z}^k \succeq 0, \quad \forall k
\end{aligned}
$$

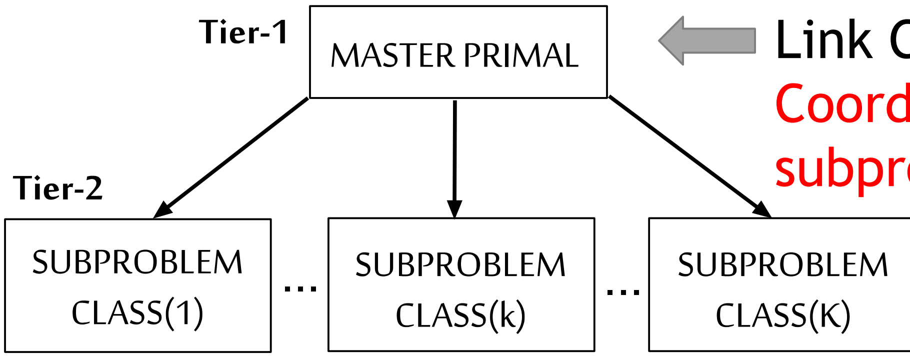
Two-Tier Design





Two-Tier Decomposition

Primal Decomposition



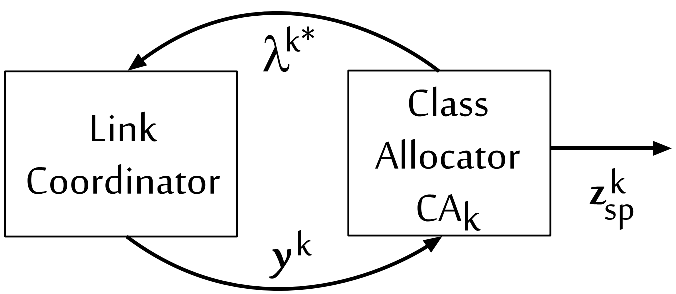
Link Coordinator

Coordinates all the subproblems

Class Allocator

Solves independently

Message-Passing



step size : β

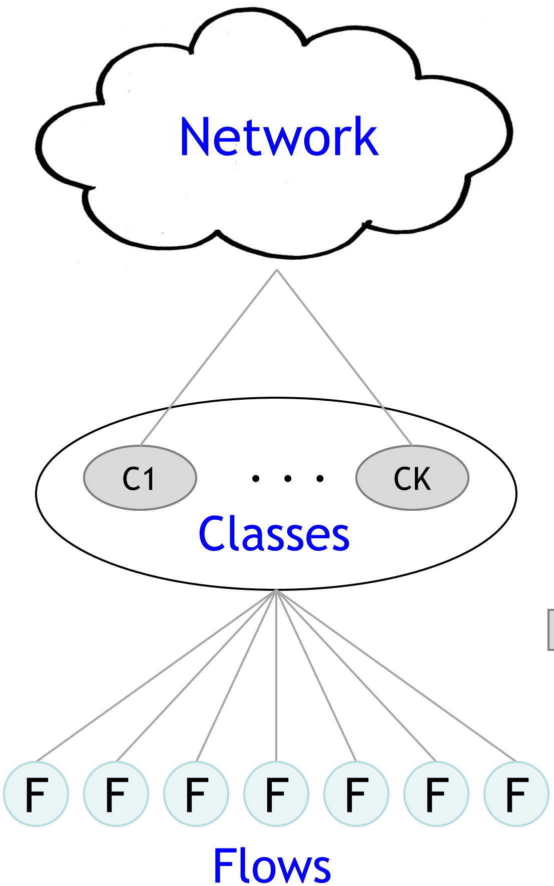
$\mathbf{y}^k$: Aggregate bandwidth assigned to class k

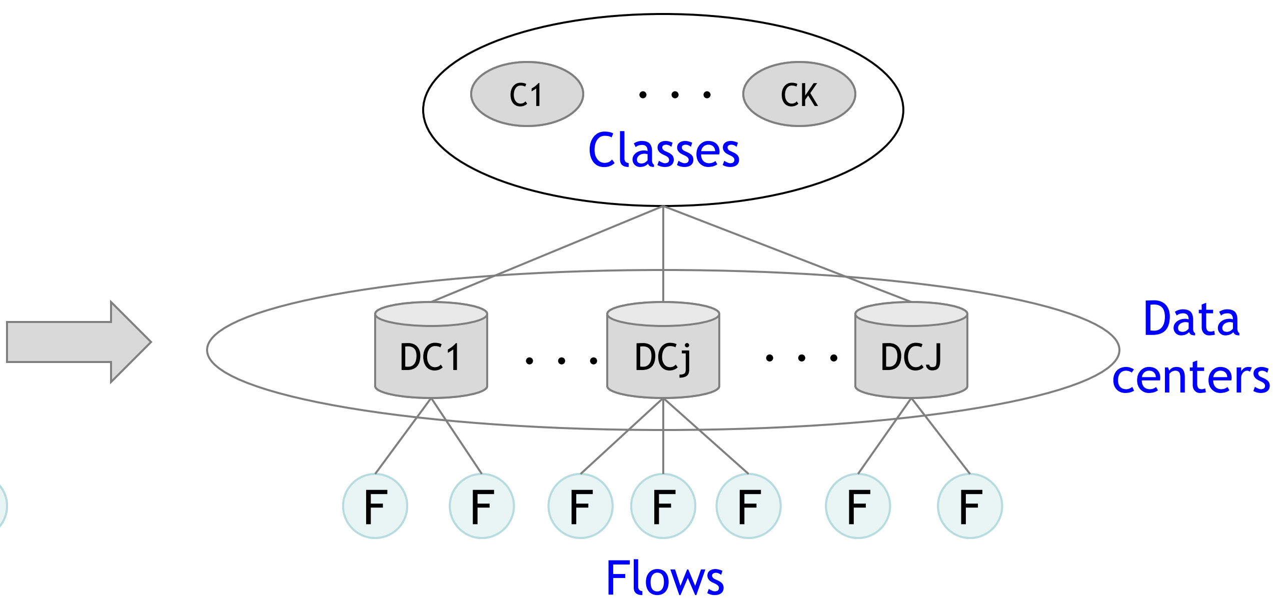
$\boldsymbol{\lambda}^{k^*}$: Optimal subgradient of CLASS(k)

Three-Tier Design

Why another tier? (High control overhead)

- Flow of a given class may originate from any DC
- Each class allocator potentially communicates with all DCs





Three-Tier Design

One centralized entity

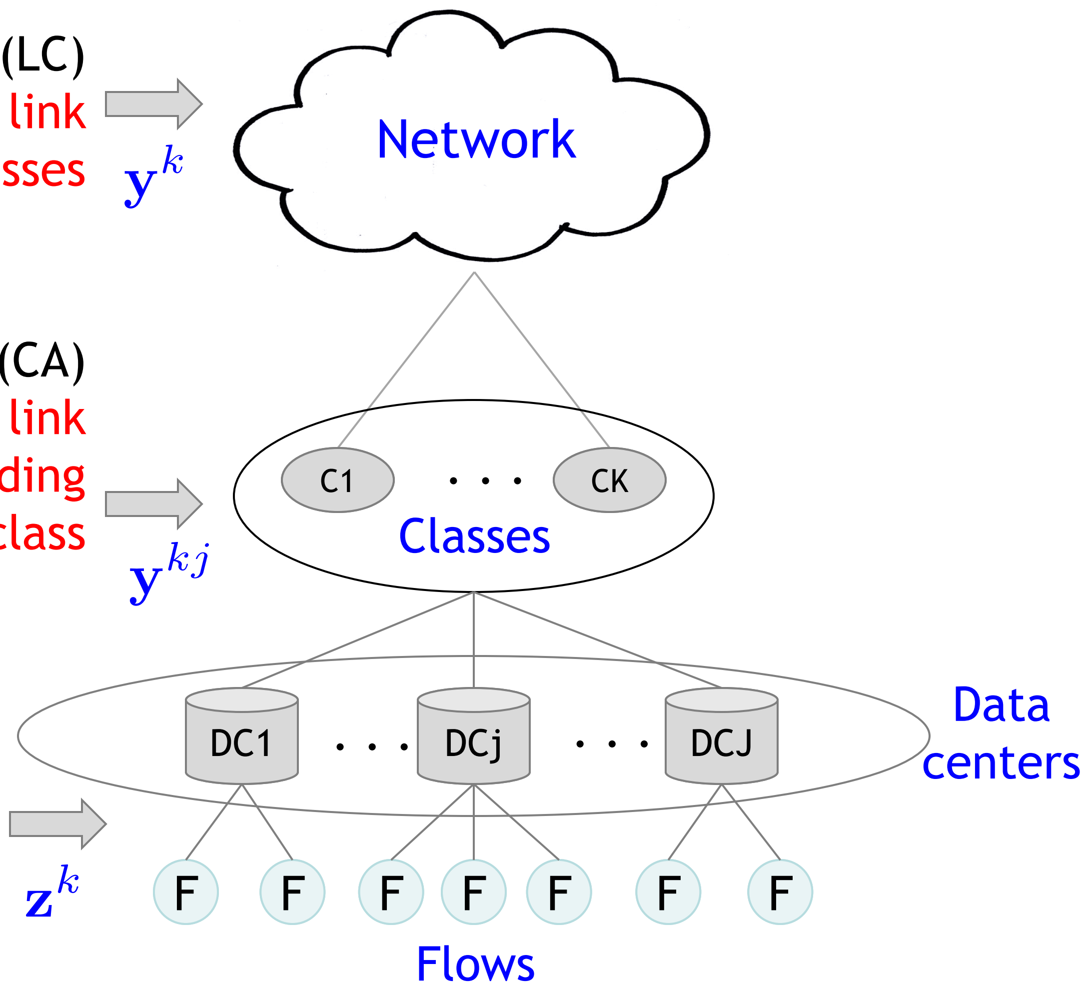
Link Coordinator (LC)
Optimizes aggregate link bandwidth across classes

One per class

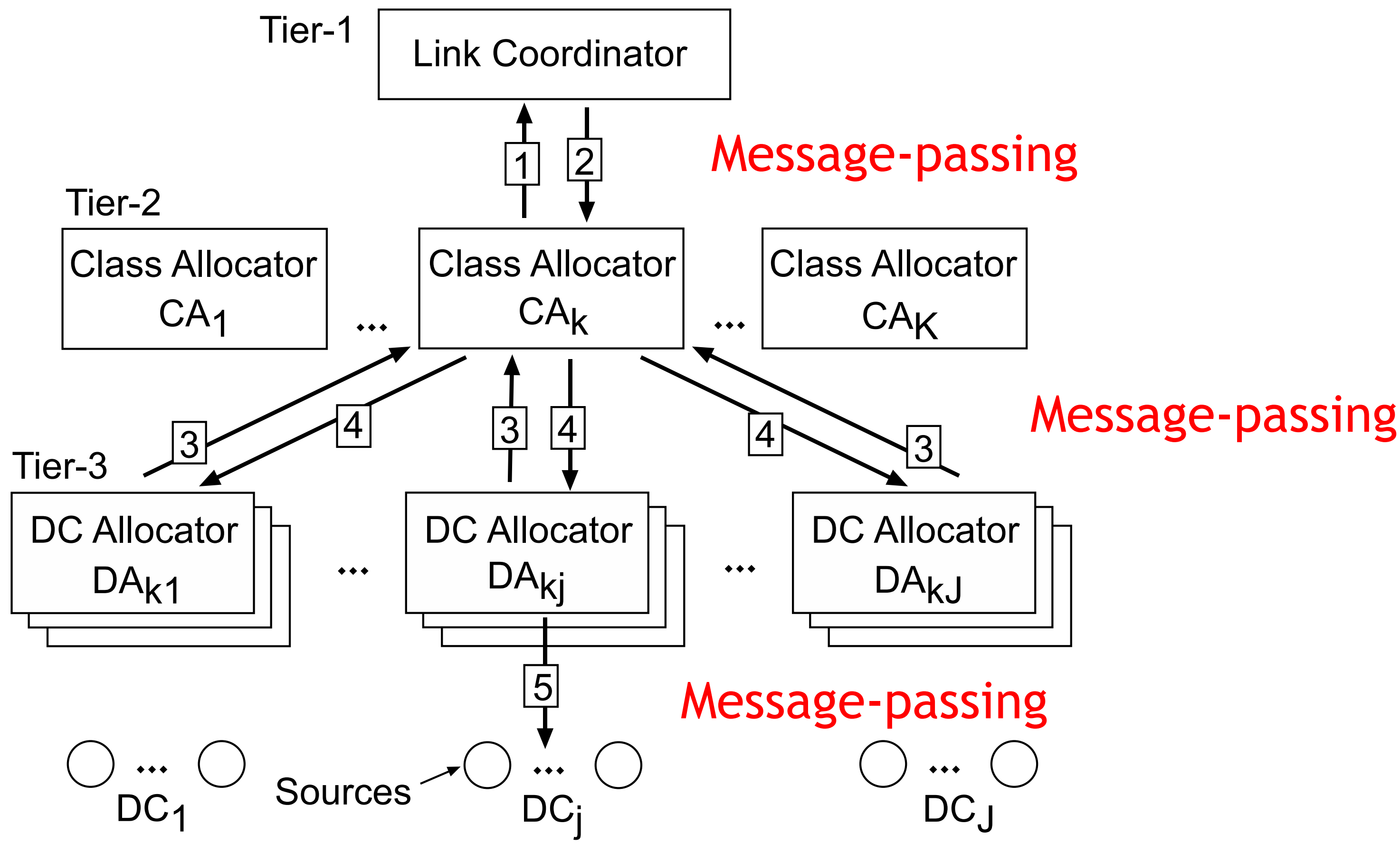
Class Allocator (CA)
Optimizes aggregate link bandwidth across DCs sending traffic in its own class

Data Center Allocator (DCA)
Optimizes sending rates across flows in its own class originating from its own DC

One per class, per DC



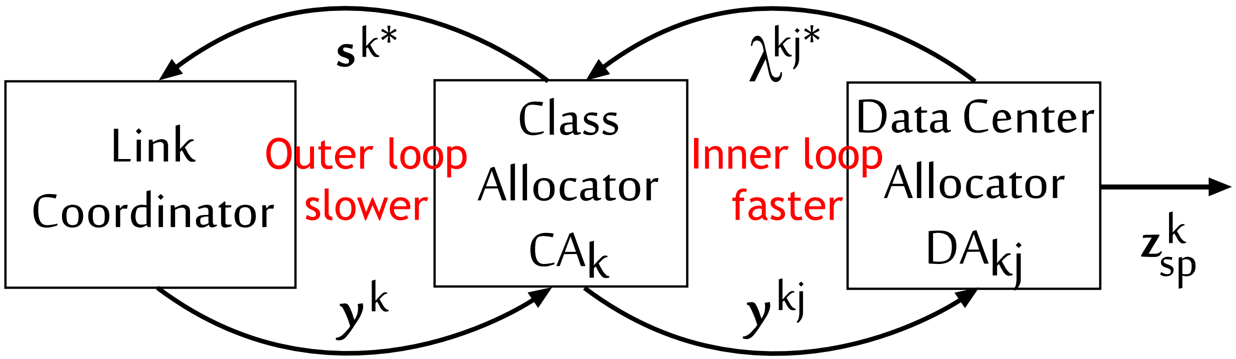
Three-Tier Design



Three-Tier Decomposition

Message-Passing

$\mathbf{s}^{k^*}$: Optimal subgradient of CLASS(k)
$\boldsymbol{\lambda}^{kj^*}$: Optimal subgradient of DATACENTER(k,j)



$\mathbf{y}^{k}$: Aggregate bandwidth assigned to class k

$\mathbf{y}^{kj}$: Aggregate bandwidth assigned to DC j sending traffic of class k

Puzzles

Please take a look at the following links:

1. http://gurmeet.net/puzzles/
2. http://www.dcg.ethz.ch/members/roger/puzzles/
3. http://research.microsoft.com/en-us/um/people/leino/puzzles.html
4. (Lateral Thinking Puzzles) http://www.thecourse.us/students/lateral_thinking.htm