Distributed Research on Emerging Applications & Machines
Department of Computational & Data Sciences
Indian Institute of Science, Bangalore

DREAM:Lab

Leveraging Cloud Computing for Big Data Platforms

Yogesh Simmhan

RWCC Workshop, *JNU, New Delhi*

22-Dec-15

Cloud Computing
End User's Perspective

- Expose capabilities
  - Compute, storage, prog. platforms, software
- as a Service
  - Using standard interfaces, REST/SOAP API
- for on-demand usage
  - Use as much or as little as you want without prior notification
- with pay as you go pricing
  - Pay only for what you use

Cloud Computing
Service Provider's Perspective

- Use commodity clusters
  - Lowers cost of acquiring hardware off the shelf, but with lower reliability
- at large data centres
  - Large volume amortizes capex, reduces opex
- located near cheap power sources
  - Electricity is typically largest opex
- managed by a Cloud fabric
  - Reduces management overhead, human intervention

Cloud Computing is Ubiquitous

- Online services
  - Hosting services, content, e.g. Facebook, web search
- Mobile apps
  - Back-end processing, e.g. WhatsApp, Maps/Directions
- Enterprises
  - Public/private Cloud model, SaaS, e.g. EMail, CRM
- Cloud Data Centres motivated Big Data platforms…

Cloud Computing for Big Data

- **MapReduce** was an outcome of *large web log data* and *Cloud data centres* at Google
- Designed for "slow" networks
  - Ethernet: Medium latency & bandwidth
- Designed for "Scale out"
  - More numbers of slower machines vs. one fast machine
- Designed to fail
  - Commodity servers and disks have lower reliability

Cloud Computing for Big Data: Map Reduce/Hadoop

- Designed for “slow” networks
  - Blocks of data rather than small messages
  - Synchronized boundaries
- Designed for “Scale out”
  - Distributed file system for cumulative I/O bandwidth
  - Map tasks are trivially parallelizable
- Designed to fail
  - Write state to disks for recovery
  - Tasks can be restart if slow/failed

Cloud Computing for Big Data Platforms

Volume
- MapReduce/Hadoop, Apache Spark
- NoSQL, HBase, Hive

Velocity
- Stream Processing, Storm, Spark Streaming
- **Complex Event Processing**

Variety
- **Graph processing**, Giraph, GraphX, GraphLab
- Deep learning, unstructured analytics, Semantic Web

Big Data Platforms Designed for Clouds

- Clouds…or Commodity Clusters
  - **Commodity clusters**: Commodity infrastructure
  - **Clouds**: Commodity infrastructure, ***on-demand elasticity & pricing***, ***centralized data centre***, massive scale-out, virtualized

- What are the unique challenges & opportunities of **Clouds** for Big Data?

Elasticity for Distributed Graph Processing

Distributed Graph Processing

- Sources of massive data: *petascale simulations, high throughput devices, Internet, scientific applications.*
- New challenges for analysis: *data sizes, heterogeneity, uncertainty, data quality, temporal variance*

Bioinformatics

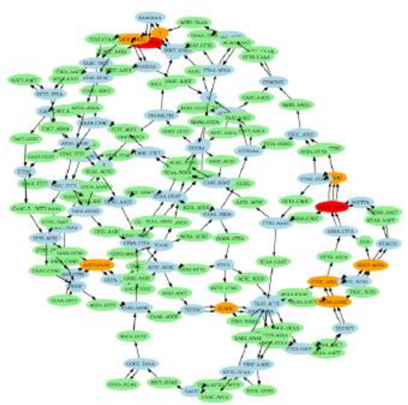
Problem: Genome & haplotype assembly, Expression Analysis
Challenges: data quality
Graph problems: Eulerian paths, MaxCut, String graphs

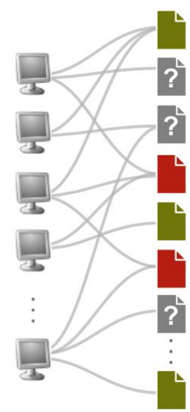
Cybersecurity

Problem: Detecting anomalies and bad actors
Challenges: scale, real-time
Graph problems: belief propagation, community analysis

Social Informatics

Problem: Discover emergent communities, spread of info.
Challenges: new analytics routines, uncertainty in data.
Graph problems: clustering, shortest paths, flows.





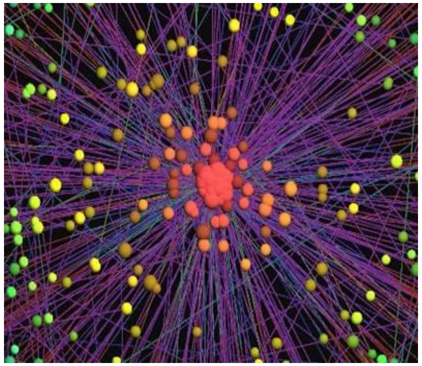
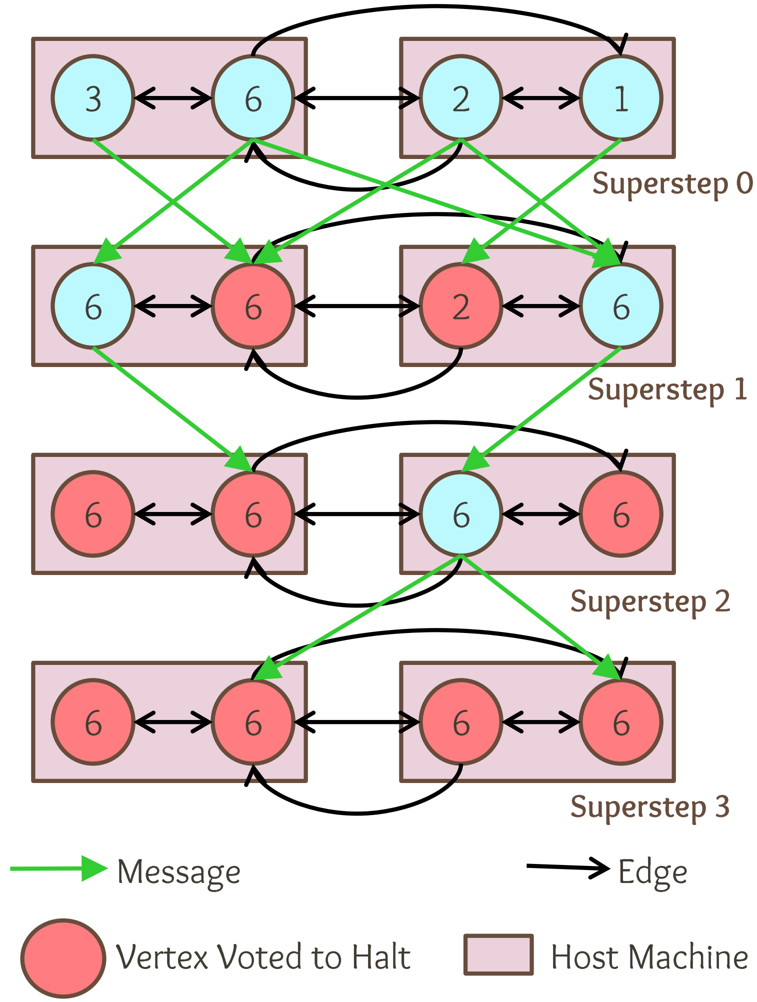


Image sources: (1) Mihai Pop, Carl Kingsford www.cbcb.umd.edu/ (2) Chau et al. In *SIAM Data Mining* (2011) (3) www.visualComplexity.com

Distributed Graph Programming Model

- **Vertex-centric Model**
  - Logic written for a single vertex
  - Execution as series of synchronized *supersteps*
- Vertices *partitioned* across multiple hosts
- Message passing between vertices. *Messages delivered* at superstep boundaries.
- *Parallelism* at vertex level
- E.g. *Google Pregel, Apache Giraph*



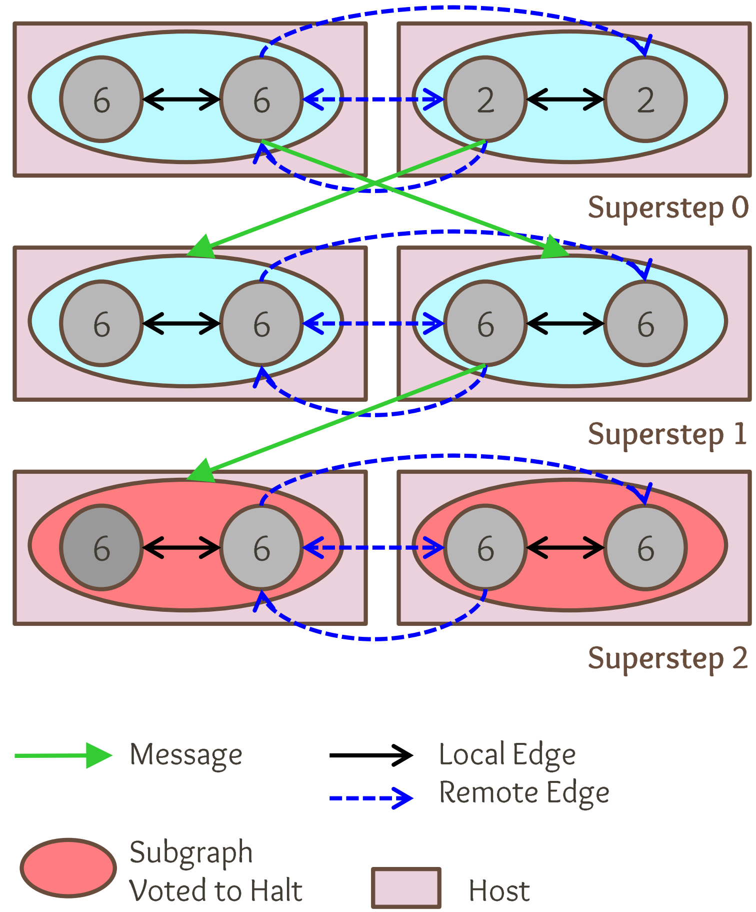
Max Vertex Value using Vertex-centric

→ But, large communication cost, more time to converge

Malewicz, Grzegorz, et al. "Pregel: a system for large-scale graph processing." ACM SIGMOD 2010.

Distributed Graph Programming Model

- Subgraph-centric Model
  - **Subgraph**: Weakly Connected Component (WCC) *within a partition*
- Logic written for a subgraph
  - Message passing between subgraphs
  - Parallelism at subgraph level
- Less communication cost, ~faster convergence
- *E.g.* ***GoFFish****, Blogel, Giraph++*



Max Vertex Value using Subgraph-centric

Y. Simmhan, et al., Goffish: A sub-graph centric framework for large-scale graph analytics, EuroPar, 2014.
Yan, Da, et al. Blogel: A block-centric framework for distributed computation on real-world graphs, VLDB 2014

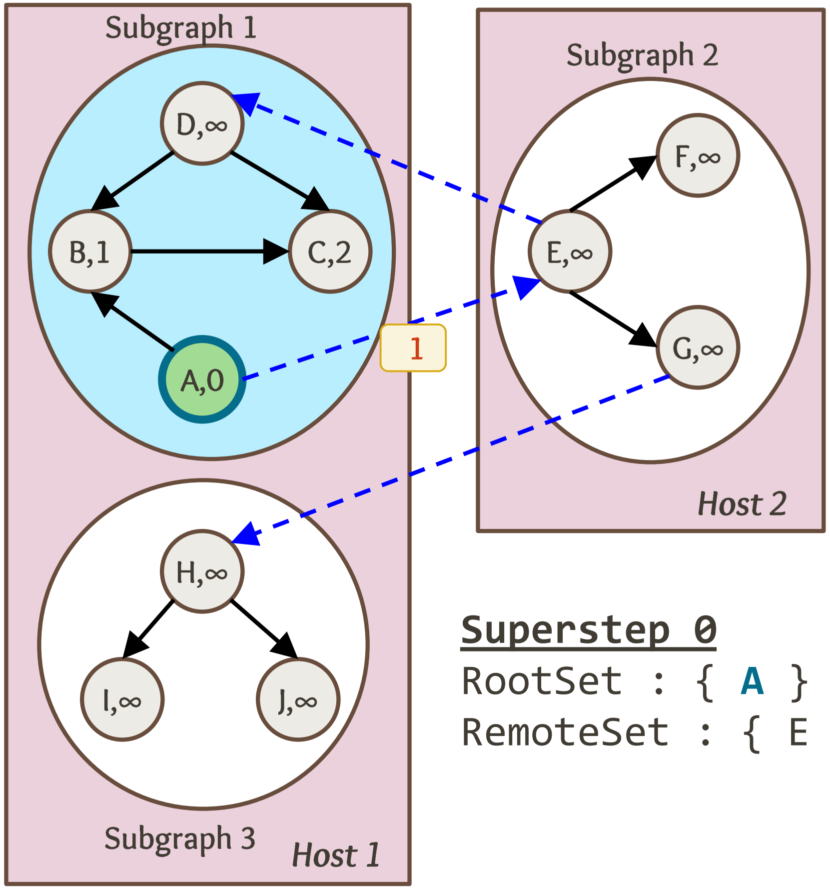
Vertex-centric Graph Processing
PageRank*

```
public void compute(Vertex<Long, Double, Float> vertex,
      Iterable<Double> messages) throws IOException {

  if (getSuperstep() >= 1) { // update my PR from remote msgs
      double sum = 0;
      for (double m : messages)  sum += m.value;
      double vertexValue = 0.15f/vertexCount() + 0.85f * sum;
      vertex.value = vertexValue;
  }

  if (getSuperstep() < MAX_SUPERSTEPS) { // send my PR
      long edges = vertex.getNumEdges();
      sendMessageToAllEdges(vertex, vertex.value / edges);
  } else
      vertex.voteToHalt();
}
```

*Apache Giraph PageRank Code

Subgraph-centric Graph Processing
Djikstras / SSSP *(Step 0)*

```
Compute (<Messages> M_arr){
  if Superstep == 0
    dist[v] =  ∞  ∀ v
    if source is present
      Rootset = {source}
      dist[source] = 0
  else
    for each message in M_arr
      if dist[m.vertex] < m.value
          dist[m.vertex] = m.value
      Rootset <- Updated vertices

  Run Dijkstra's on RootSet
  Send Messages to Remote Vertices
  VoteToHalt()
}
```



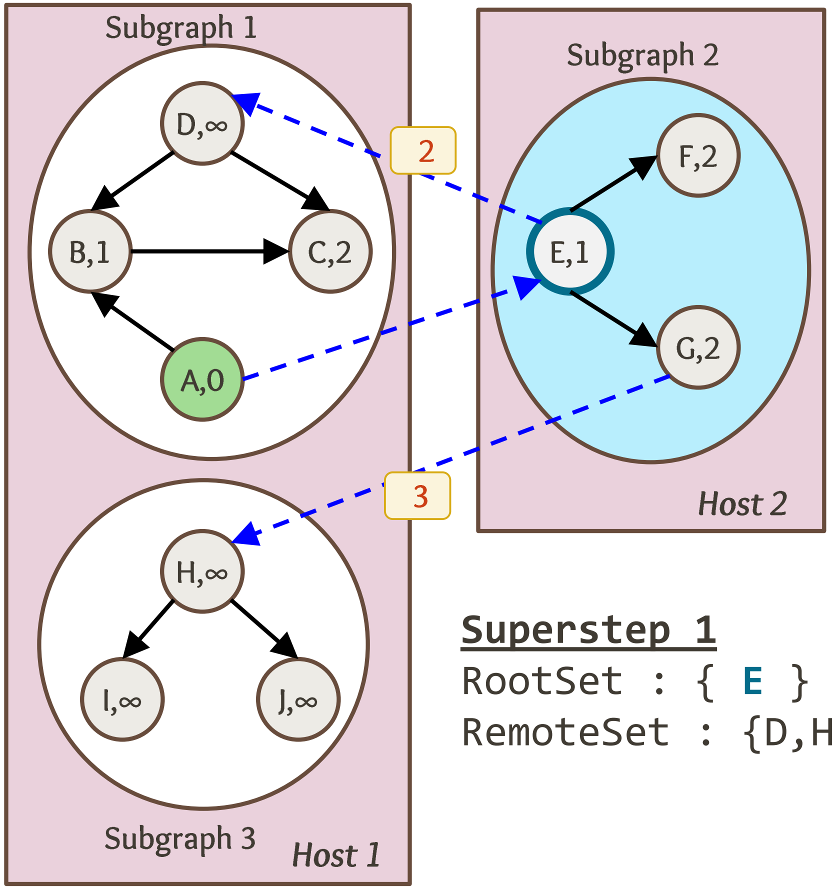
Superstep 0
RootSet : { **A** }
RemoteSet : { E }

Note: Edges are Unweighted

Y. Simmhan, et. Al., "Goffish: A sub-graph centric framework for large-scale graph analytics," EuroPar, 2014.

Subgraph-centric Graph Processing
Djikstras / SSSP *(Step 1)*

```
Compute (<Messages> M_arr){
  if Superstep == 0
    dist[v] =  ∞  ∀ v
    if source is present
      Rootset = {source}
      dist[source] = 0
  else
    for each message in M_arr
      if dist[m.vertex] < m.value
          dist[m.vertex] = m.value
      Rootset <- Updated vertices

  Run Dijkstra's on RootSet
  Send Messages to Remote Vertices
  VoteToHalt()
}
```

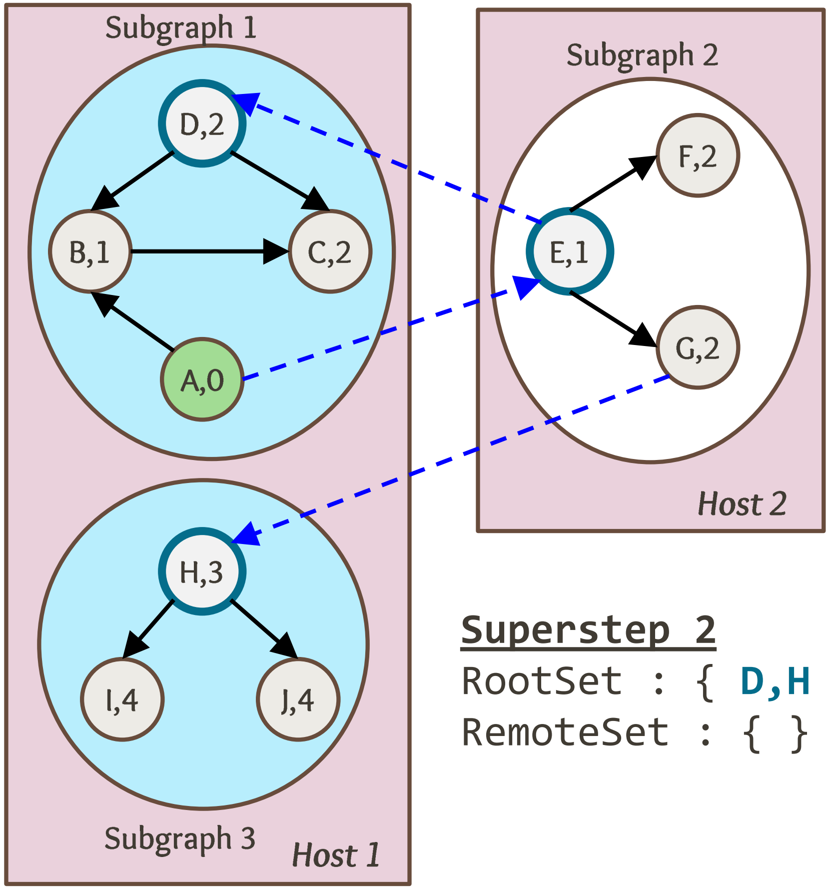


Note: Edges are Unweighted

Superstep 1
RootSet : { E }
RemoteSet : {D,H}

Subgraph-centric Graph Processing
Djikstras / SSSP *(Step 2)*

```
Compute (<Messages> M_arr){
  if Superstep == 0
    dist[v] =  ∞  ∀ v
    if source is present
      Rootset = {source}
      dist[source] = 0
  else
    for each message in M_arr
      if dist[m.vertex] < m.value
          dist[m.vertex] = m.value
      Rootset <- Updated vertices

  Run Dijkstra's on RootSet
  Send Messages to Remote Vertices
  VoteToHalt()
}
```



Superstep 2
RootSet : { **D,H** }
RemoteSet : { }

Note: Edges are Unweighted

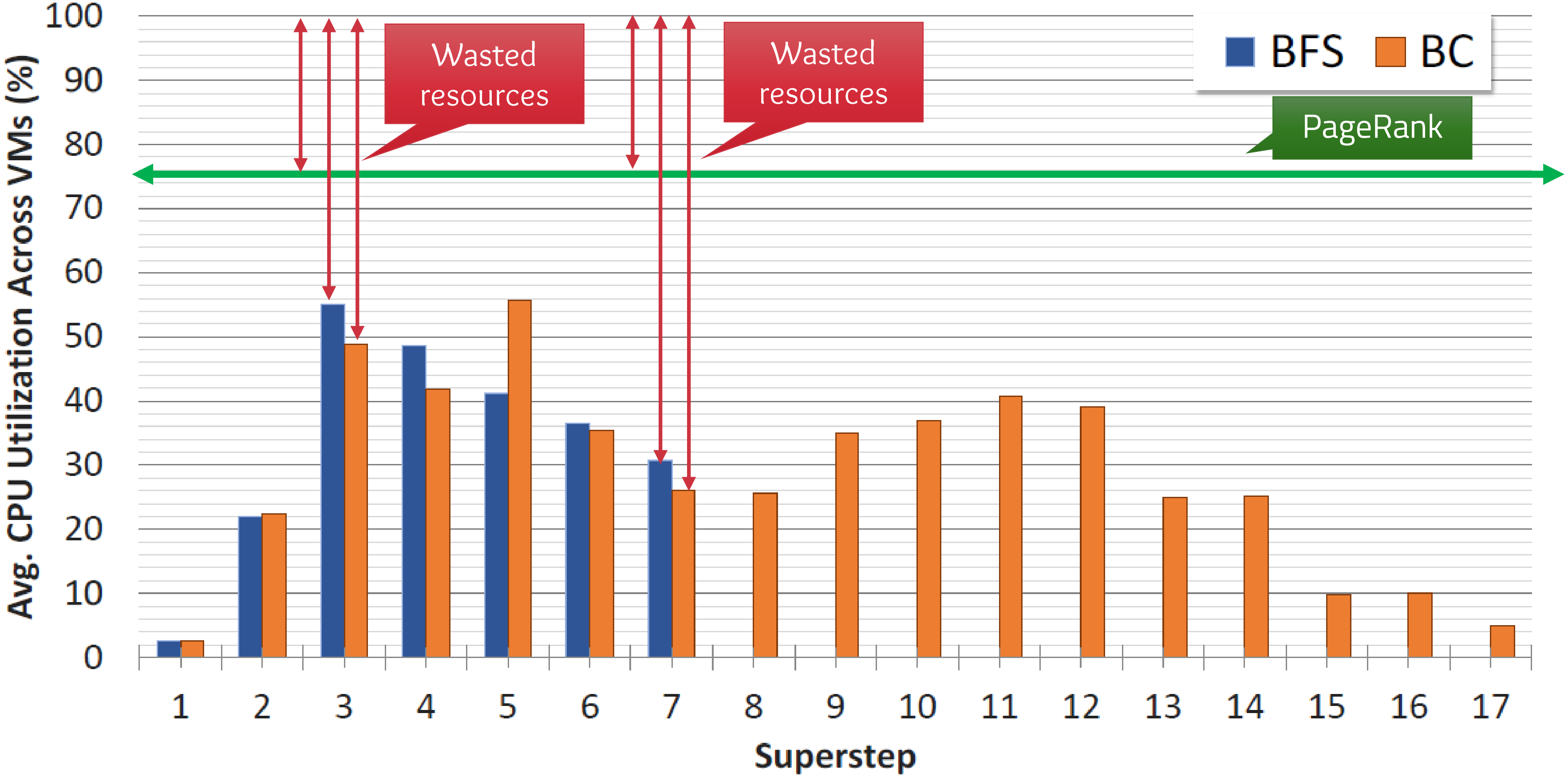
Stationary vs. Non-stationary Graph Algorithms

- Stationary algorithms, *e.g. PageRank*
  - Same amount of work done in each iteration, by each worker
  - Uniform resource utilization
- Non-Stationary algorithms, *e.g. SSSP*
  - Different amount of work done in each iteration, by each worker
  - Variable resource utilization
  - Over-allocation (or) Under-performance

Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis, "Mizan: a system for dynamic load balancing in large-scale graph processing," in EuroSys, 2013

CPU Usage Across Iterations

- Orkut Graph (3M vertices, 234M edges)
- 40 cores, 5 machines



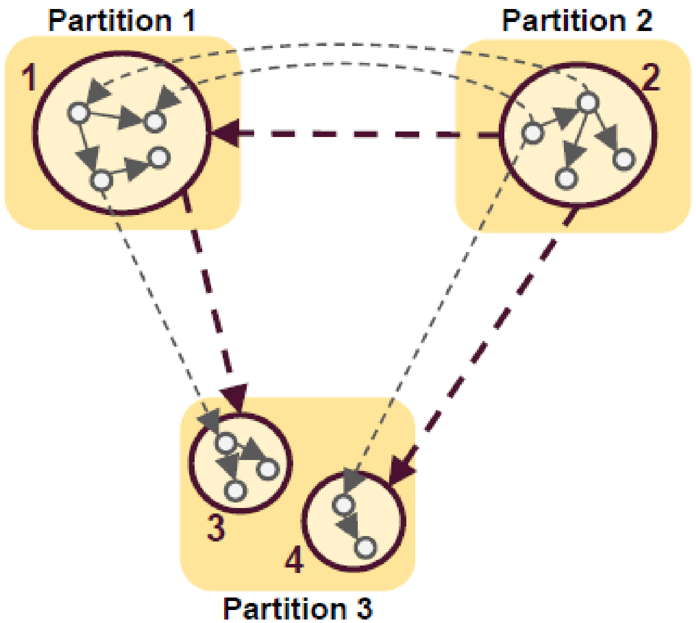
Elastic Resource Allocation for Non-stationary Distributed Graph Algorithms, Ravikant Dindokar and Yogesh Simmhan, Under review, 2015

Clouds Elasticity for Graph Processing

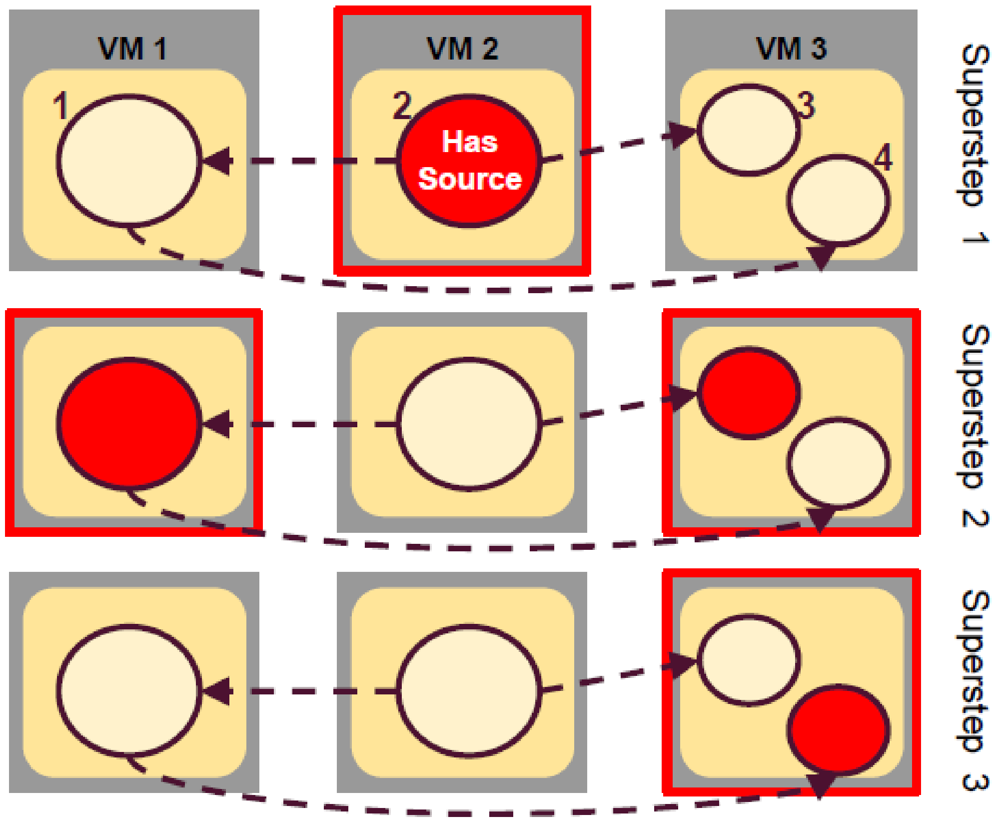
- Graph processing load changes with each iteration
- Under-utilization ➲ Higher cost to utility

- Challenge: Can we use “Elasticity” to increase utilization?
  1. Find the load in an iteration ➲ *Find the partitions of the graph that are active*
  2. Use only as many VMs as needed ➲ *Place active partitions on live VMs*

Predicting Active Partitions: **BFS**



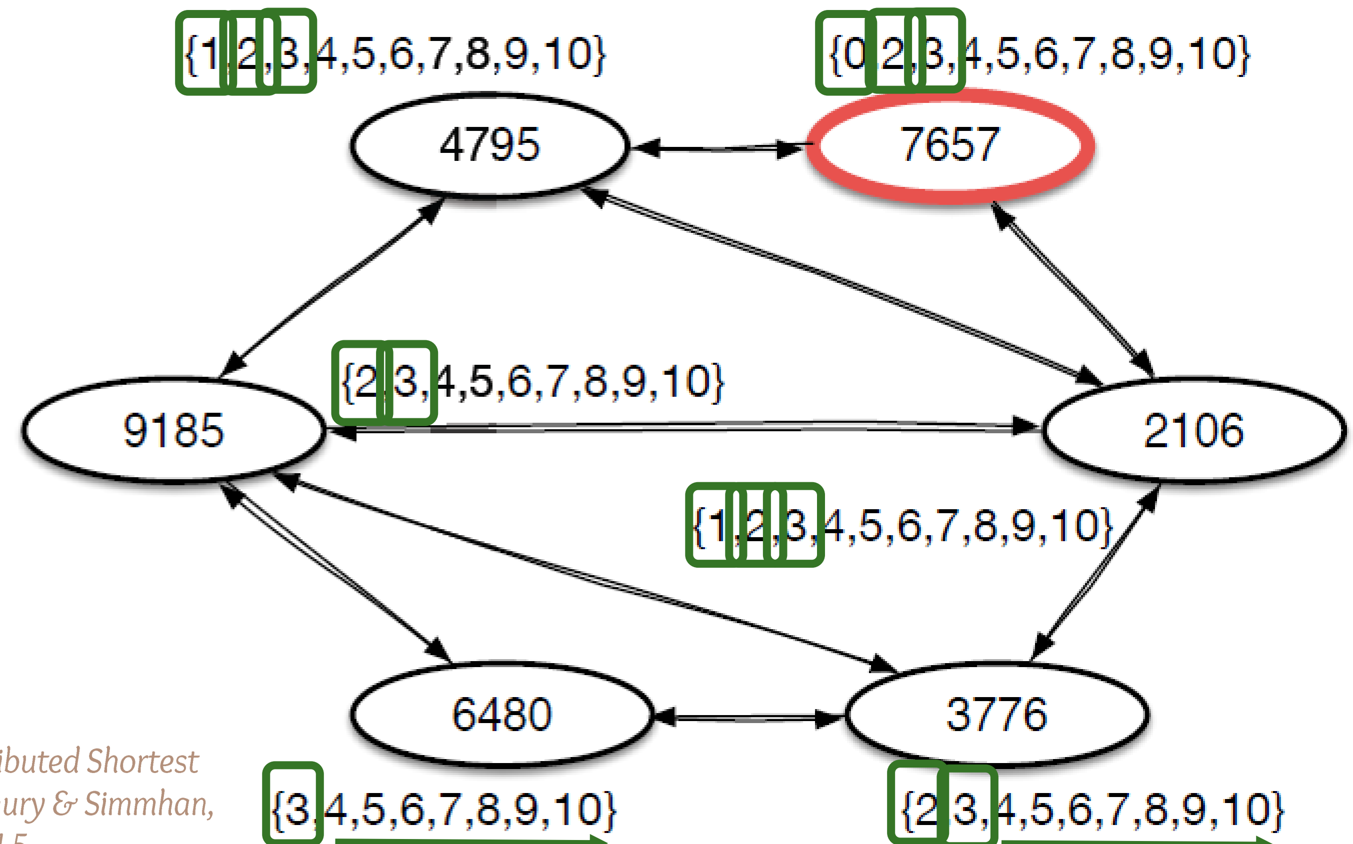
1) **Graph** with 13 vertices/13 edges *(small gray circles & lines)* divided into **three Partitions** *(yellow rectangles)*.
2) **Four Subgraphs** *(large purple circles, labeled 1-4)* identified within the partitions.
3) **Meta-graph** formed has four subgraphs as meta-vertices & three meta-edges *(purple dashed line)* connecting them.



■ One partition placed in each VM for execution ■ **BFS** from source vertex in Subgraph 2 causes only VM 2's usage in Superstep 1; VM 1 & 3 are idle ■ Subgraphs 1 & 3 are active in Superstep 2, causing VMs 1 & 3 to be used, and VM 2 is idle ■ Subgraph 4 active in Superstep 3 causes only VM 3 to be used

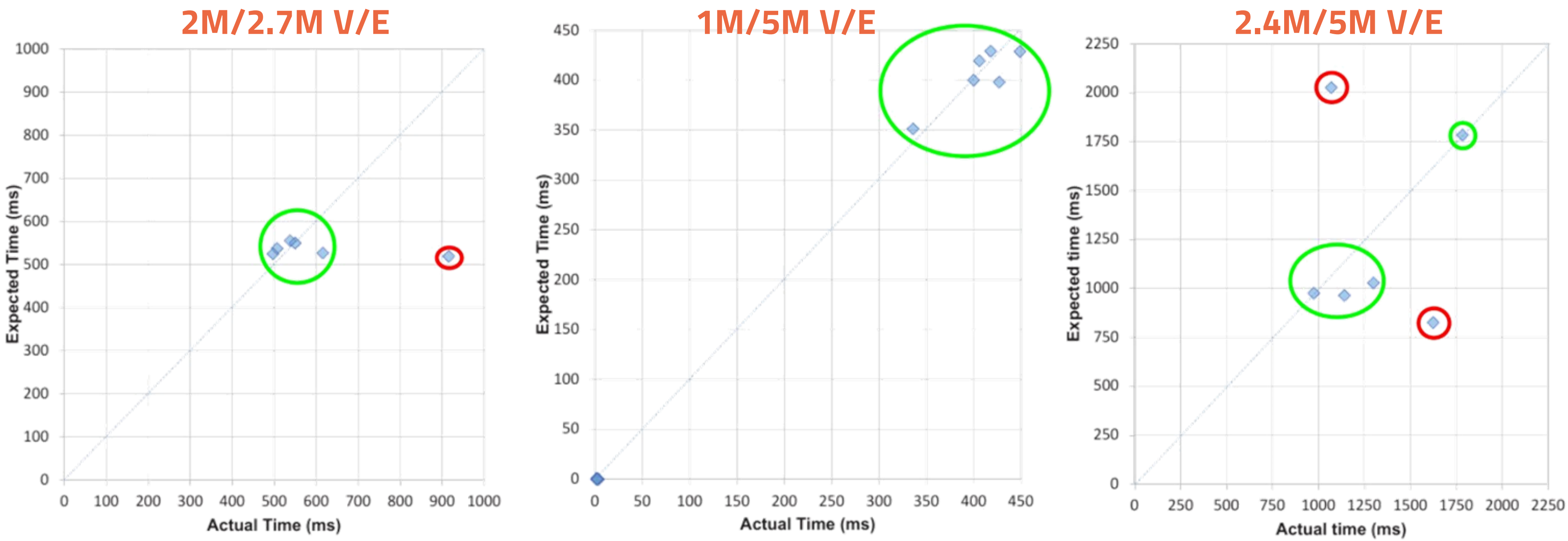
Meta-Graphs for Algorithm Modelling

- "Meta-Vertices" are subgraphs
- Iteration on which a meta-vertex in active



Analysis of Subgraph-centric Distributed Shortest Path Algorithm, Dindokar, Choudhury & Simmhan, IPDPS ParLearning workshop, 2015

Dijkstra's Prediction: Expected vs. Actual



(a) California Road Network (CARN) (b) Web Google (WEBG) (c) Wiki Talk (WIKI)

- *Dijkstra's called exactly at superstep corresponding to traversal depth.*
- *Expected and observed time complexity matches closely.*
- Outliers: Subgraph with source and subgraph with large number of incoming messages

**Expected time is normalized by multiplying it by a constant α*
**Plot showing only non tiny subgraphs (|V| > 100)*

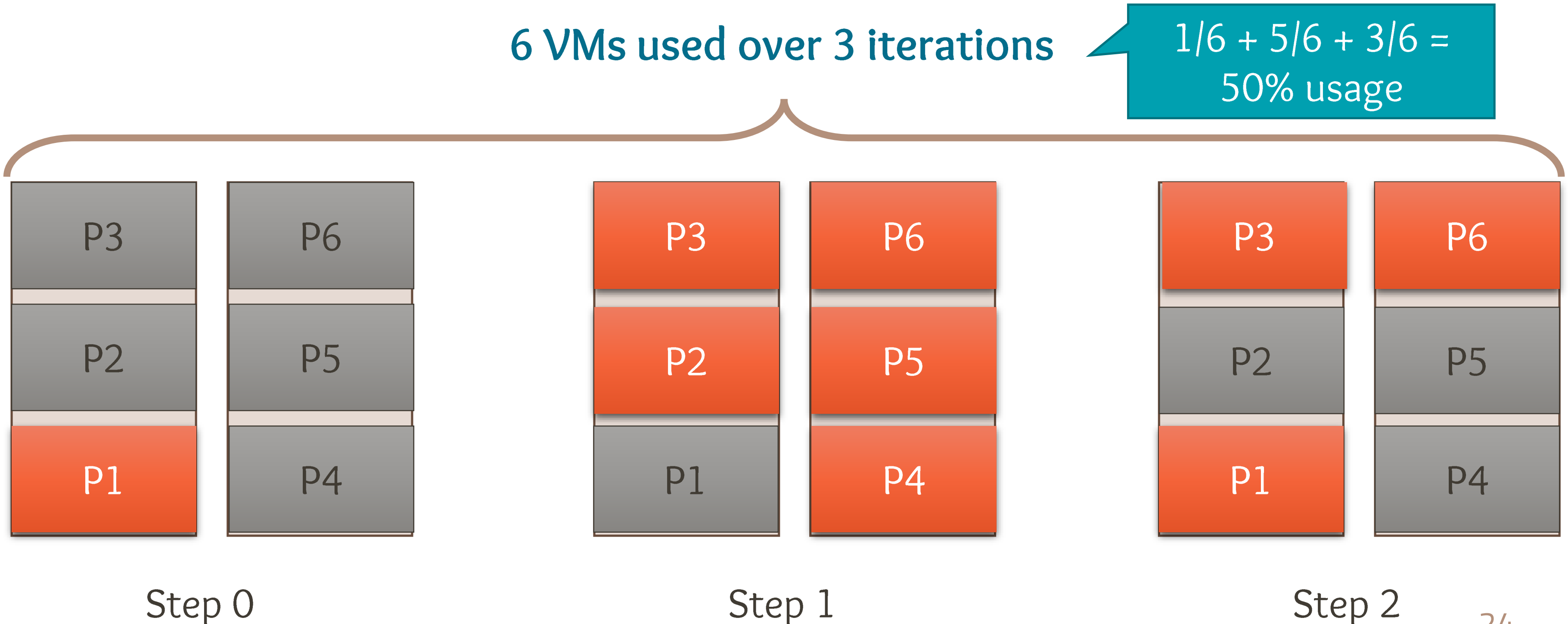
Graph Partition Placement on VMs

- How we can reduce the ***overall monetary cost*** for running the graph algorithm
- with ***minimal impact on the makespan*** of the algorithm,
- using ***partition placement strategies*** on elastic VMs
- based on their ***activation schedule*** across supersteps,
- as compared to a traditional hashing of partitions onto a static set of VMs.

Elastic Resource Allocation for Non-stationary Distributed Graph Algorithms, Ravikant Dindokar and Yogesh Simmhan, Under review, 2015

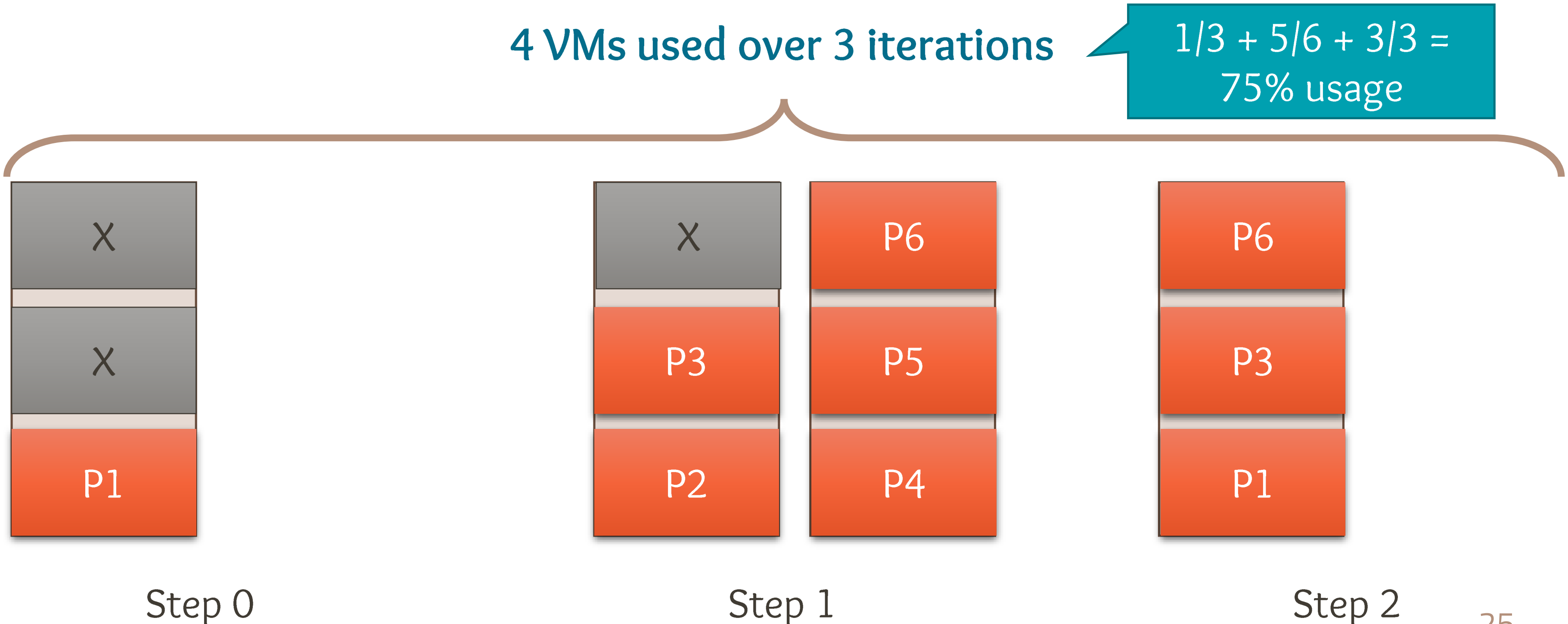
Default Strategy, *Static Placement*

- Partitions distributed across fixed count of VMs
- Uniform number of partitions per VM
  - Load balanced for stationary algorithm
- Partitions placement is static across iterations



First Fit Decreasing (FFD)

- Number of VMs per iteration depends on load
  - Elastic scale out and in
- Pack active partitions on available VMs
  - Bin packing/knapsack problem
- Partition movement cost between iterations

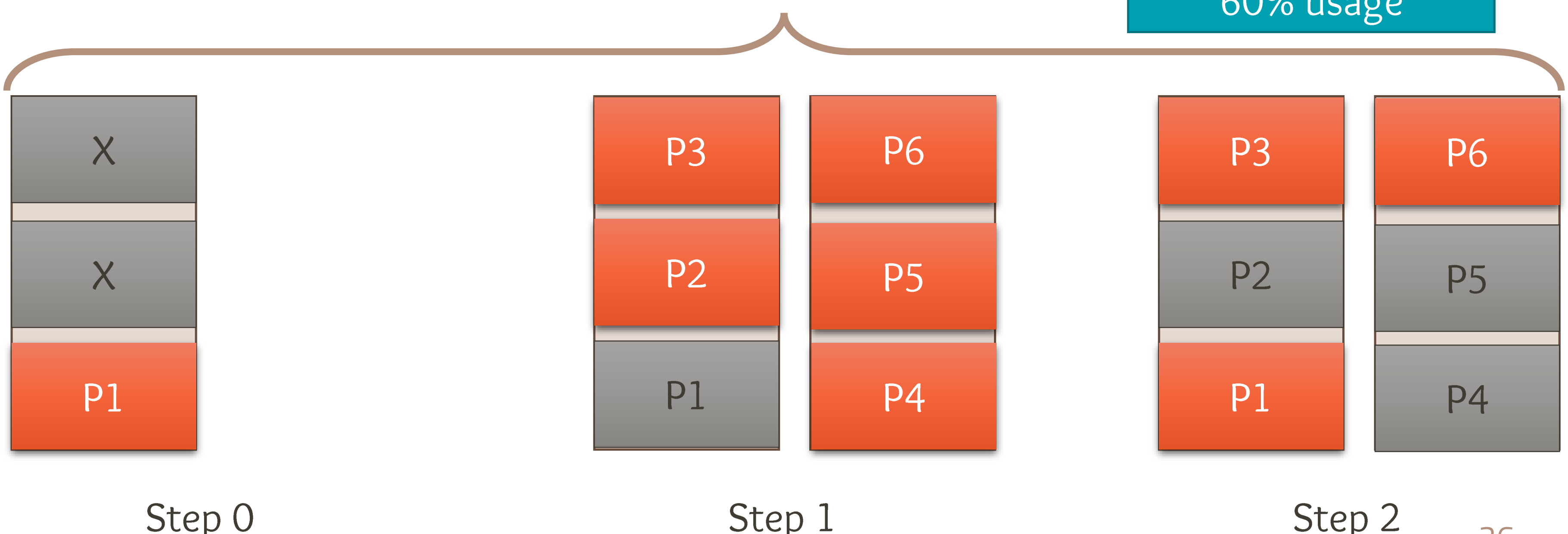


Max Fit with Pinning (MF/P)

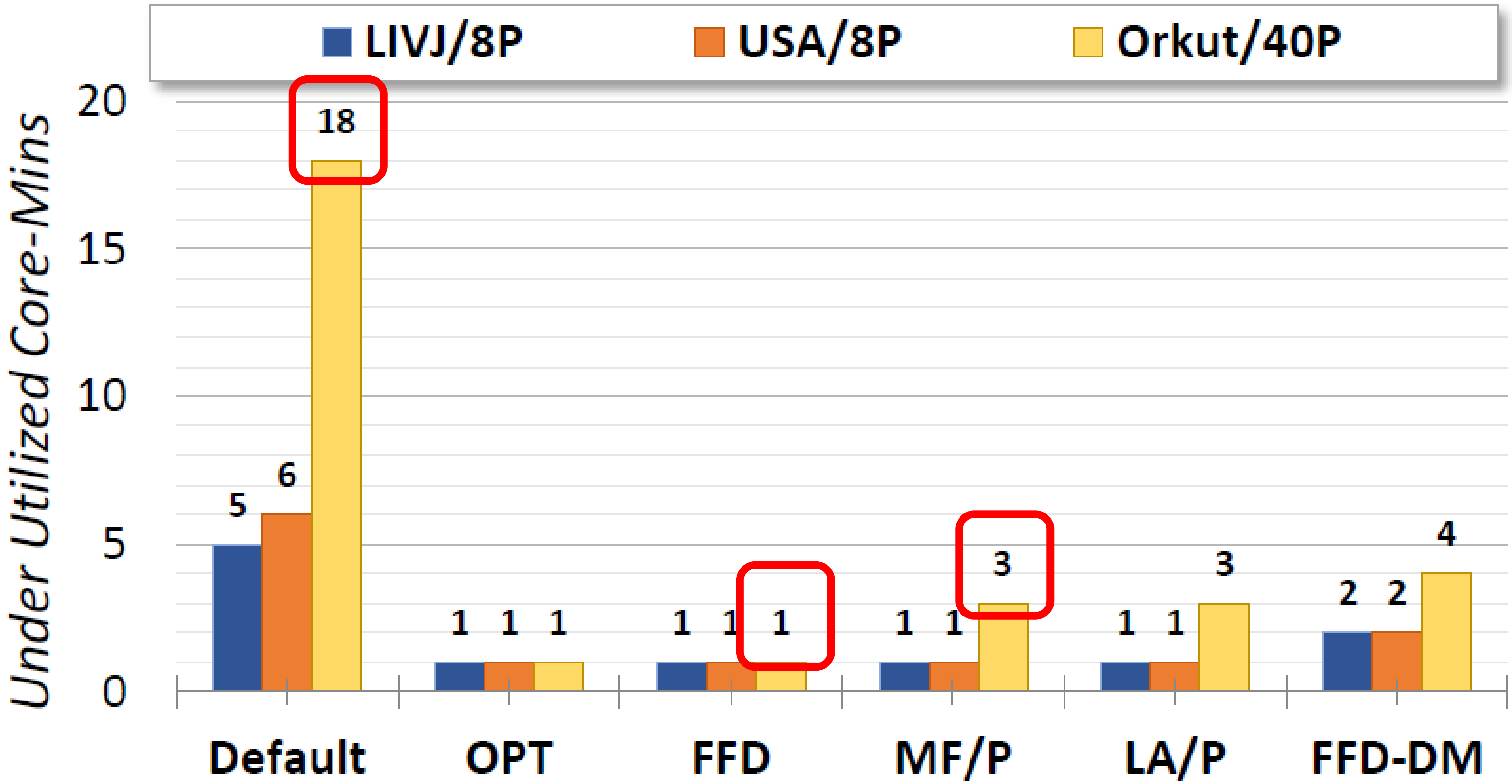
- Number of VMs per iteration depends on load
  - Partial elastic scale out and in
- Partitions placement is static *once pinned*
  - No movement cost
  - Load distribution can be unbalanced

5 VMs used over 3 iterations

$1/3 + 5/6 + 3/6 =$
60% usage



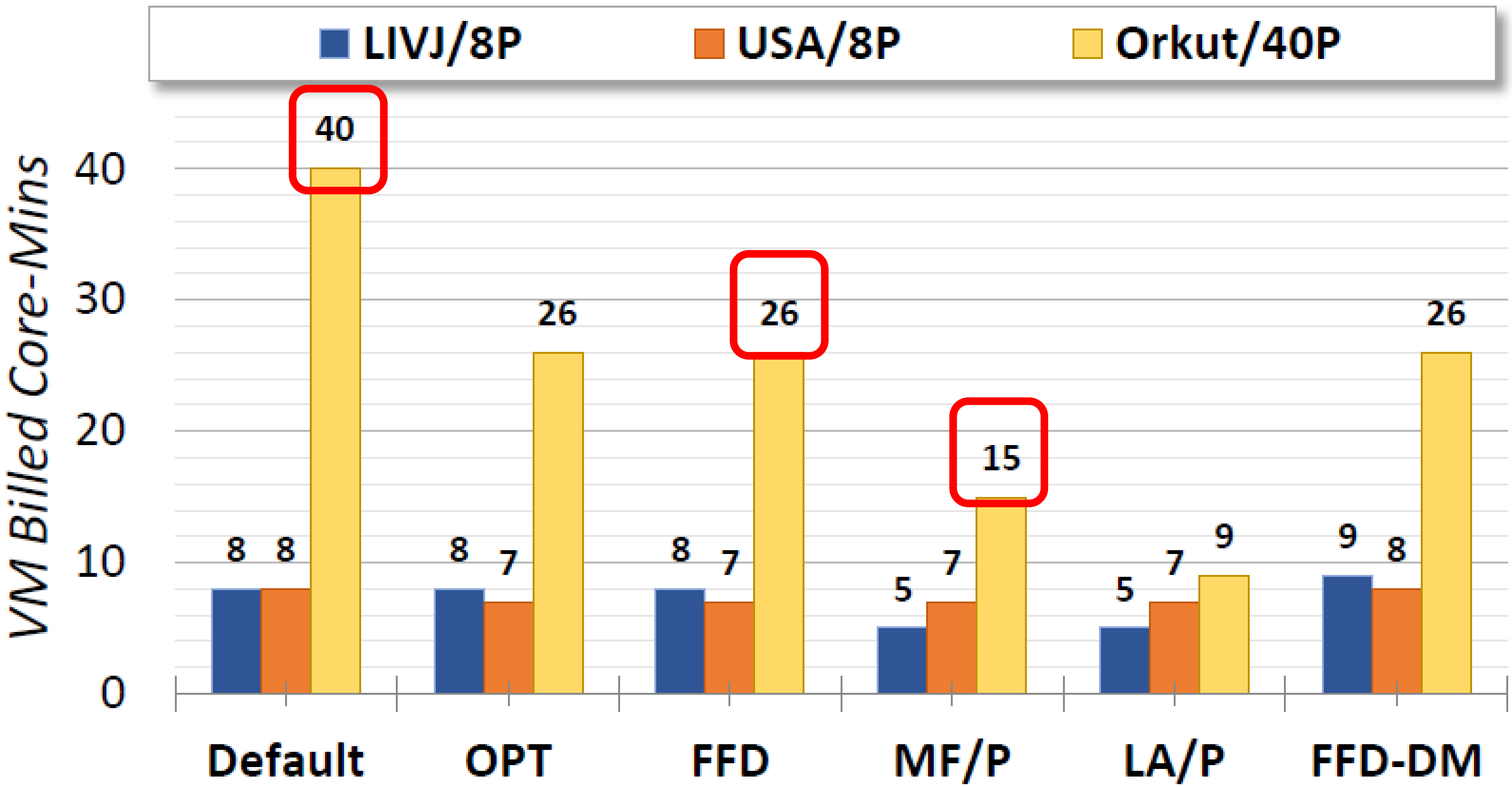
Under-utilization *(less is better)*



(g) Under Utilization for BFS

Elastic Resource Allocation for Non-stationary Distributed Graph Algorithms, Ravikant Dindokar and Yogesh Simmhan, Under review, 2015

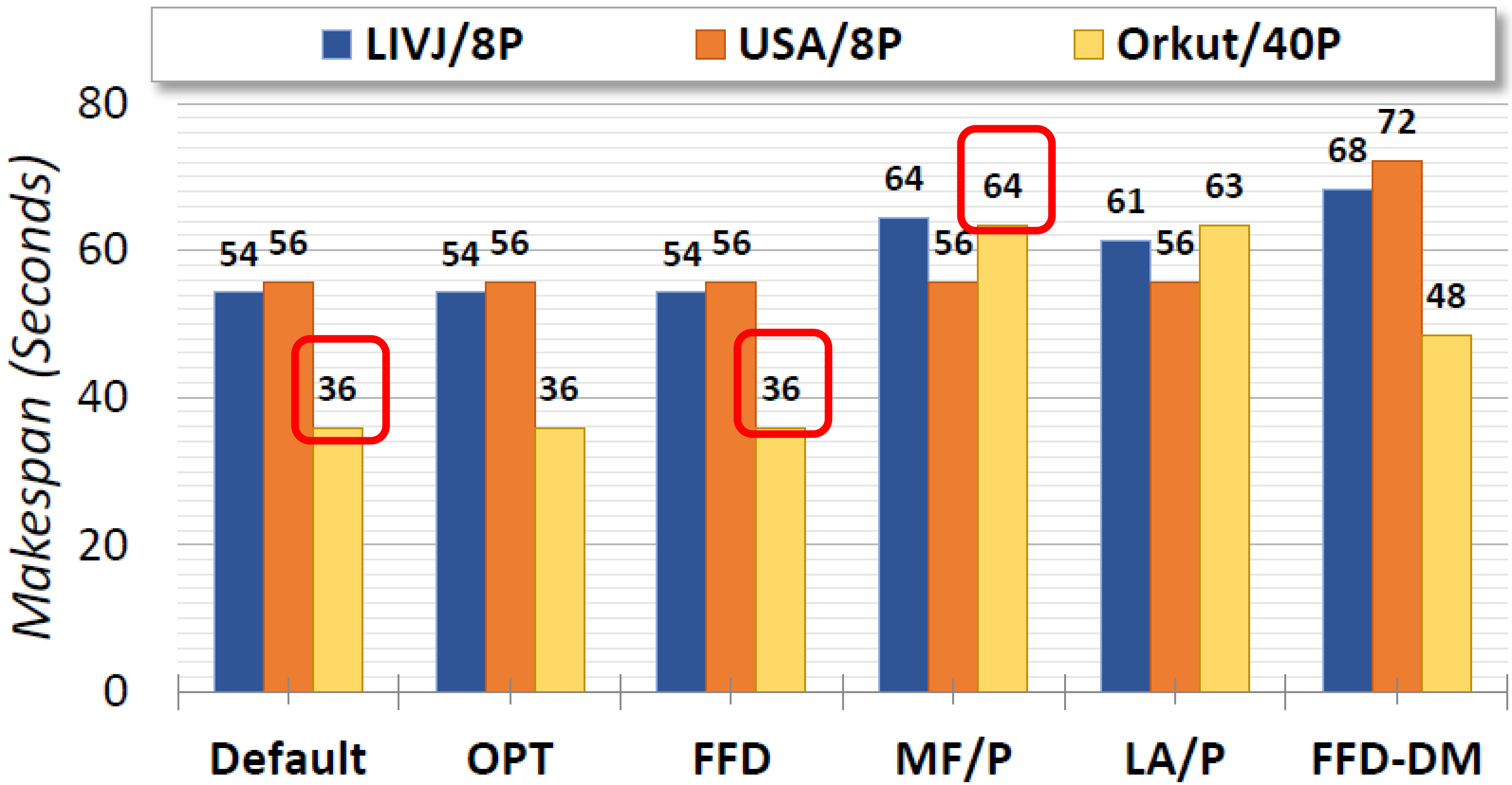
Monetary Cost *(less is better)*



(d) Core-Mins for BFS

Elastic Resource Allocation for Non-stationary Distributed Graph Algorithms, Ravikant Dindokar and Yogesh Simmhan, Under review, 2015

Makespan *(less is better)*



(a) Makespan for BFS

Elastic Resource Allocation for Non-stationary Distributed Graph Algorithms, Ravikant Dindokar and Yogesh Simmhan, Under review, 2015

Edge+Cloud for Event Processing in IoT

“Fog Computing”?

Big Data in the Age of IoT

PS1 telescope

Large Hadron Collider

Illumina NGS @ IISc

Bluetooth Mote @ IISc

Smart Meter @ LADWP

Few Instruments, Large Data **Volume**

10^2 Sources
TB's Data
Days to Proc.

Many Devices, Volume & Velocity

10^5 Sources
GB's Data
Hours to Proc.

Numerous Sensors, High data **Velocity**

10^8 Sources
MB's Data
<Mins to Proc.

IISc Smart Water IoT Project

- Plan pumping operations for reliability
  - Avoid water running out/overflow
  - It can take 12 hrs to fill a large OHT
  - Water scarcity for several weeks in the year
- Provide safer water
  - Leakages, contamination from decades old N/W
- Reduce water usage for sustainability
  - IISc avg: 400 Lit/day, Global std: 135 Lit/day
  - Lack of visibility on usage footprint, sources
  - Rain water harvesting, Water recycling plant
- Lower the cost
  - Reduce water use & energy cost for pumping

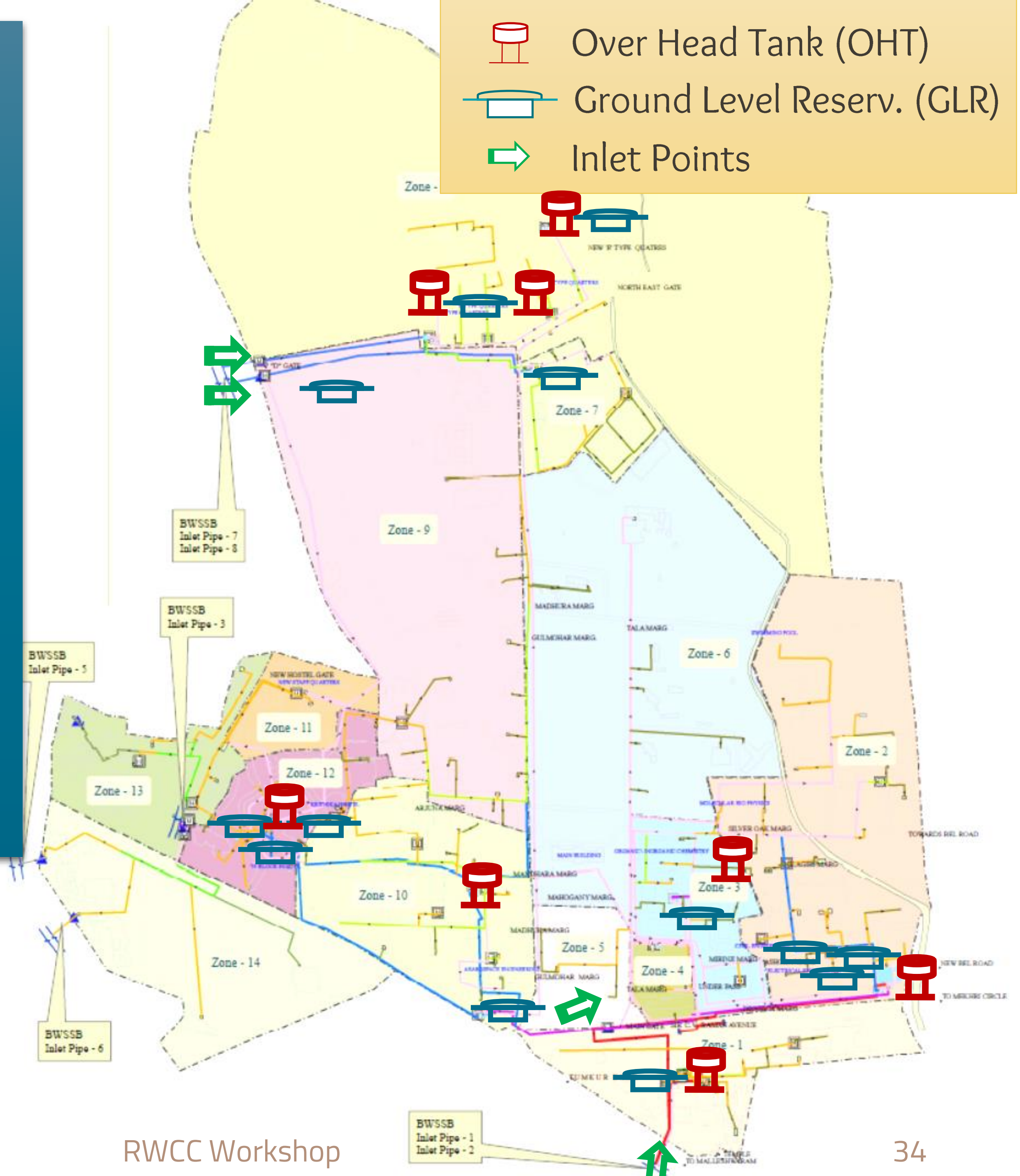
Objectives

1. *Can we use IoT & Big Data technologies to make campus "smarter"?*
   - i.e. the "infrastructure", not the people ☺
   - More efficient, reliable & safe resource delivery & management
   - Initial Case Study: **Water management**
2. *And in the process, understand the technology and improve on it?*
   - For the Indian context!

IISc Campus

- Area: 440 Acres, 8 Km Perimeter
- 50 buildings: *Office, Hotel, Residence, Stores*
- 10,000 people
- 10MW Power Consumed
- 40 Lakh Lit/Day Water Consumed

OHT	8
GLR	13
Inlet	4



Over Head Tanks (OHT)


TPH (near Mechanical)


JNT Auditorium


Chemical Stores


Opposite to CENSE

Ground Level Reservoirs (GLR)


Opposite to CENSE

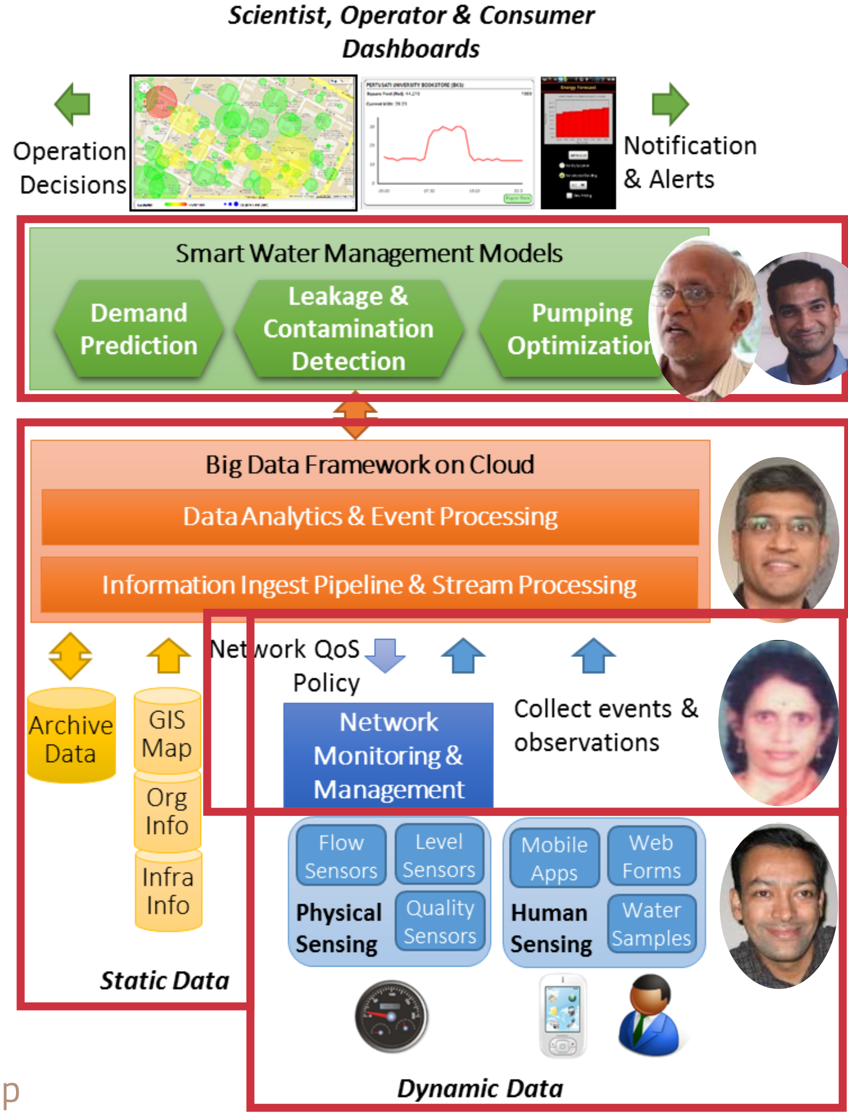

Boys Hostel


Near C Mess


Near R Block

Open, integrated & extensible **IoT Technology Stack** for Smart Campus Resource Management

1. Hybrid sensing
2. Adaptive networking
3. Realtime Analytics
4. Science-driven decision making

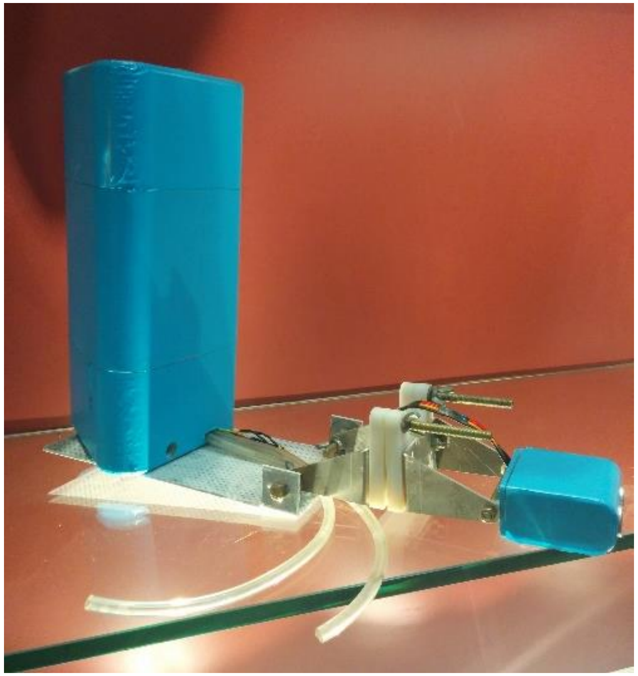


» Experts to *Close the Loop from Network to Knowledge*

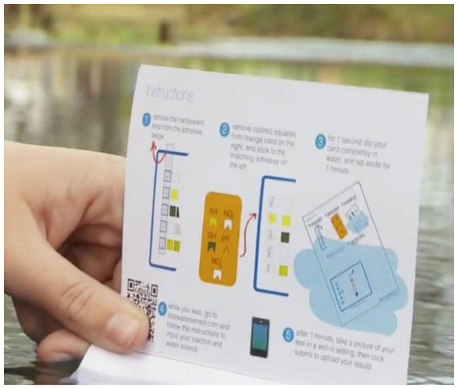
» Validate using *real-world deployment @ IISc*

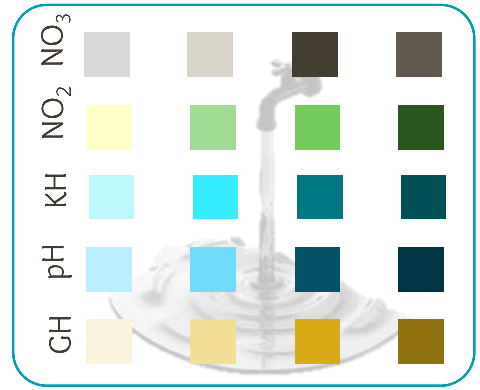
Low-Cost Sensors

- Based on commodity H/W, in-house design, QC
  - Robust to external use
  - O(min) sampling
- Water level sensors
  - Water present in OHT, GLR
  - Rate of inflow, outflow
- Water Quality
  - TDS, temperature
  - Physical, not chemical



Crowd-sourced Sensing

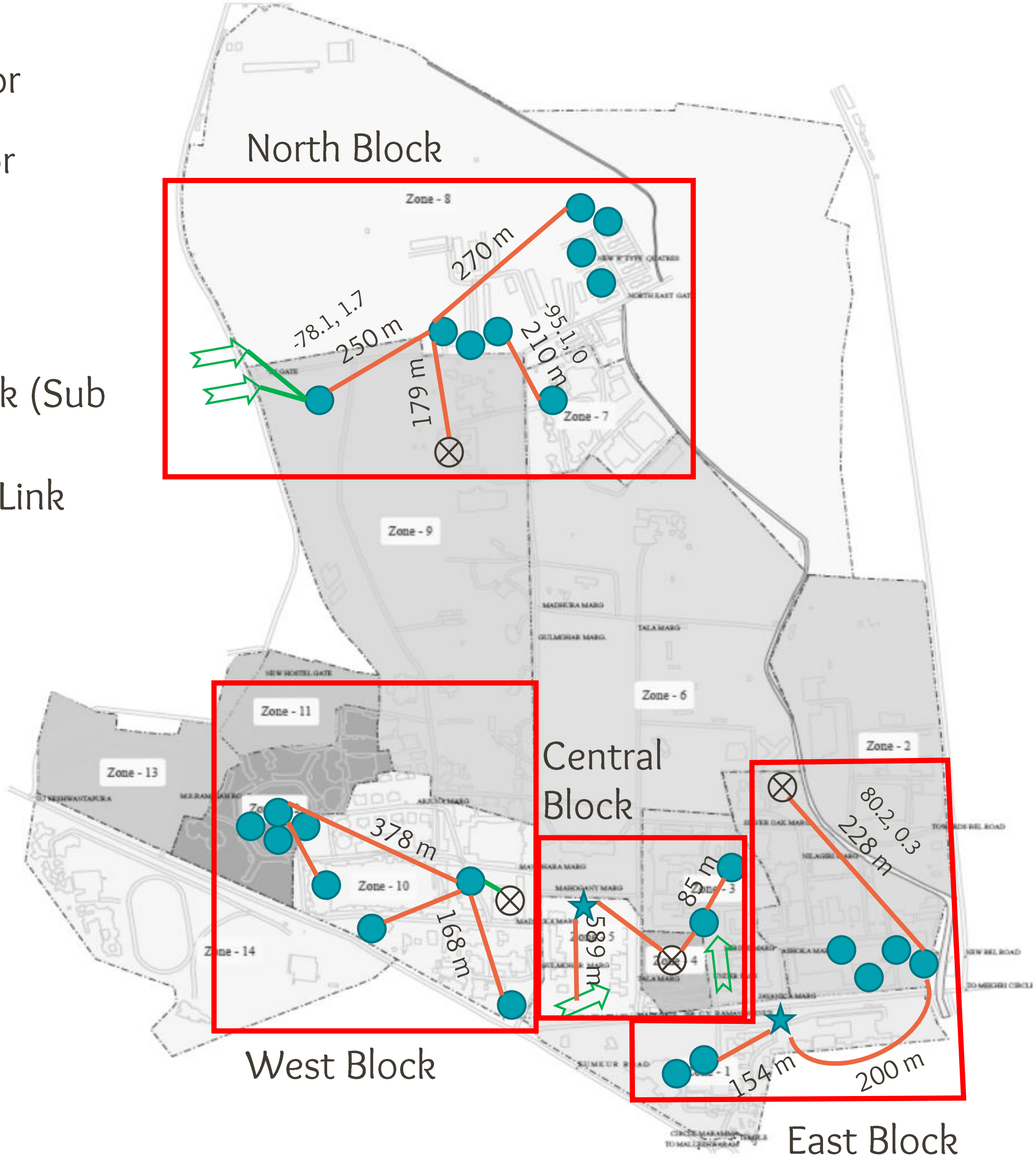


© Jose Gomez-Marquez, MIT


- **Cost-effective** quality sensing thru' Crowd Sourcing
  - Paper-based design allows diverse users to test water
  - Report via photo of card & Smart Phone App
  - A simple colorimetric, diagnostic developed by MIT to test *pH, calcium (Ca2+), magnesium (Mg2+), carbonates (CO32-), bicarbonates (HCO-3), and nitrites (NO-2 and NO-3)*
- Other ideas: App-based OCR Sensing for water meter

COMMUNICATION

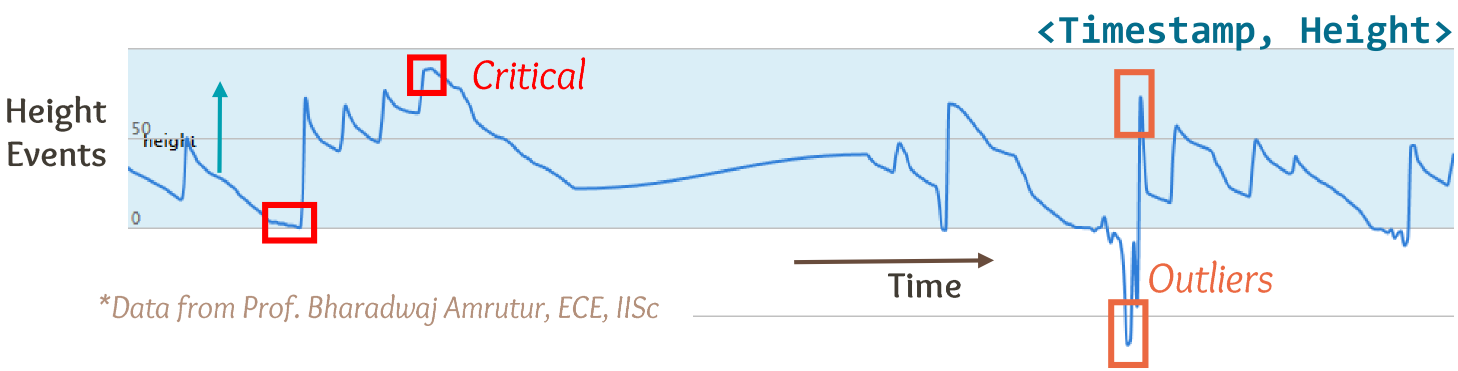
- Level Sensor
- Flow Sensor
- Relay
- Gateway
- Primary Link (Sub GHz)
- Secondary Link (Zigbee)



Fast Data Processing

Complex Event Processing (CEP)

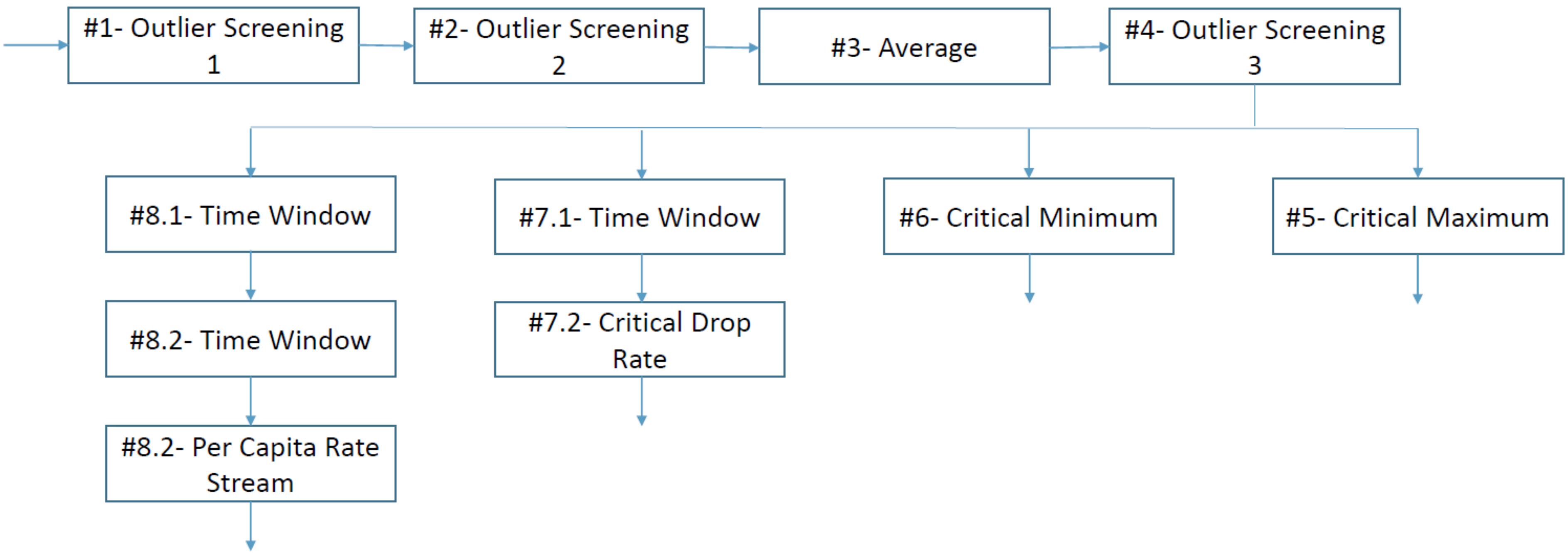
- Extract info from realtime, event sources to help ***decision-making***
  - Specify ***queries*** on situations, patterns, causal relationships
  - Online analysis of 1000's of Events per Sec



**Data from Prof. Bharadwaj Amrutur, ECE, IISc*

```
SELECT e FROM STREAM dese_oht WHERE e.height > 90%

SELECT e_hi, e_lo FROM STREAM rbccps_oht
  WHERE (e_hi.height – e_lo.height) > 5%
  WITHIN WINDOW(5mins)
```

CEP Water Analytics Pipeline



Sample CEP Queries

1. Outlier Screening 1 : Only allow valid streams in the first filtering - that are non negative and below the tank height. *from HeightEventStream [height $\geq$ 0.0 and height $\leq$ tankHeight], insert into NonOutlierStream1.*

2. Outlier Screening 2 : Only allow those streams with water height that fall within 4 times the statically computed standard deviation either way. *from NonOutlierStream1 [height $>$ -4 * stdDev and height $<$ 4 * stdDev], insert into NonOutlierStream2.*

3. Average stream : Find the average of the water height of streams in an event window length of 3 to obtain a reasonable measure of what range the height currently is at. *from NonOutlierStream2 # window.length(3), select avg(height) as avgHt, insert into AvgStream.*

Sanity Checks

Aggregation

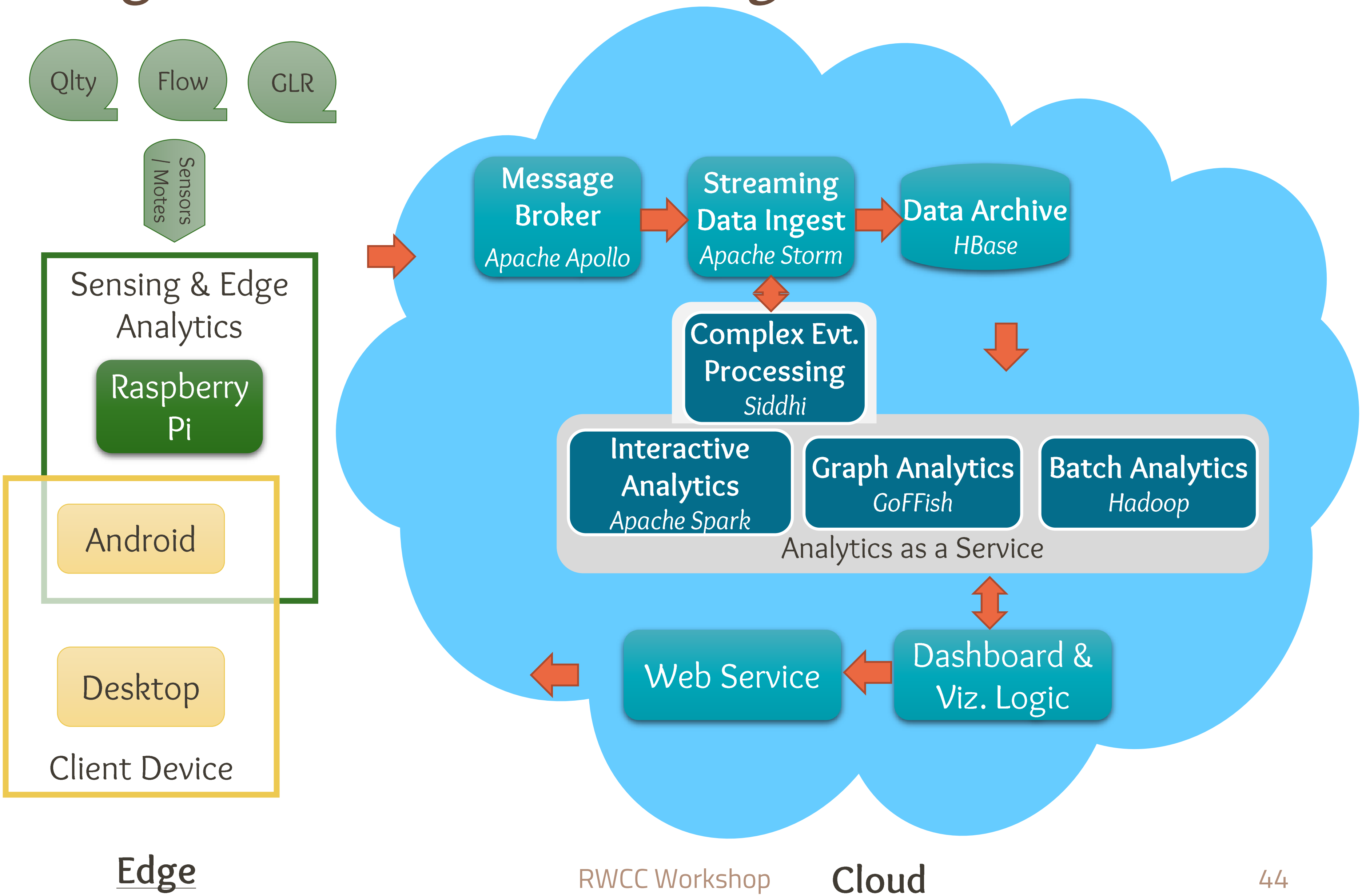
Sample CEP Queries

Analytics

5. Critical Maximum : When height of the water is above 90 of tank height, the stream is flagged as a critical stream and tied to trigger an output/action to rectify the critically high water level that may result in an overflow and thus wastage.
*from NonOutlierStream [height > 0.9 * tankHeight], insert into CriticalMaxStream.*

6. Critical Minimum : When height is below 10 of tank height, the stream is flagged as a critical stream and tied to trigger an output/action to rectify the critically low water level that may result in an underflow compared to requirements.
*from NonOutlierStream [height < 0.1 * tankHeight] , insert into CriticalMinStream.*

Big Data Platform on *Edge+Cloud*



Analytics from Edge to Cloud

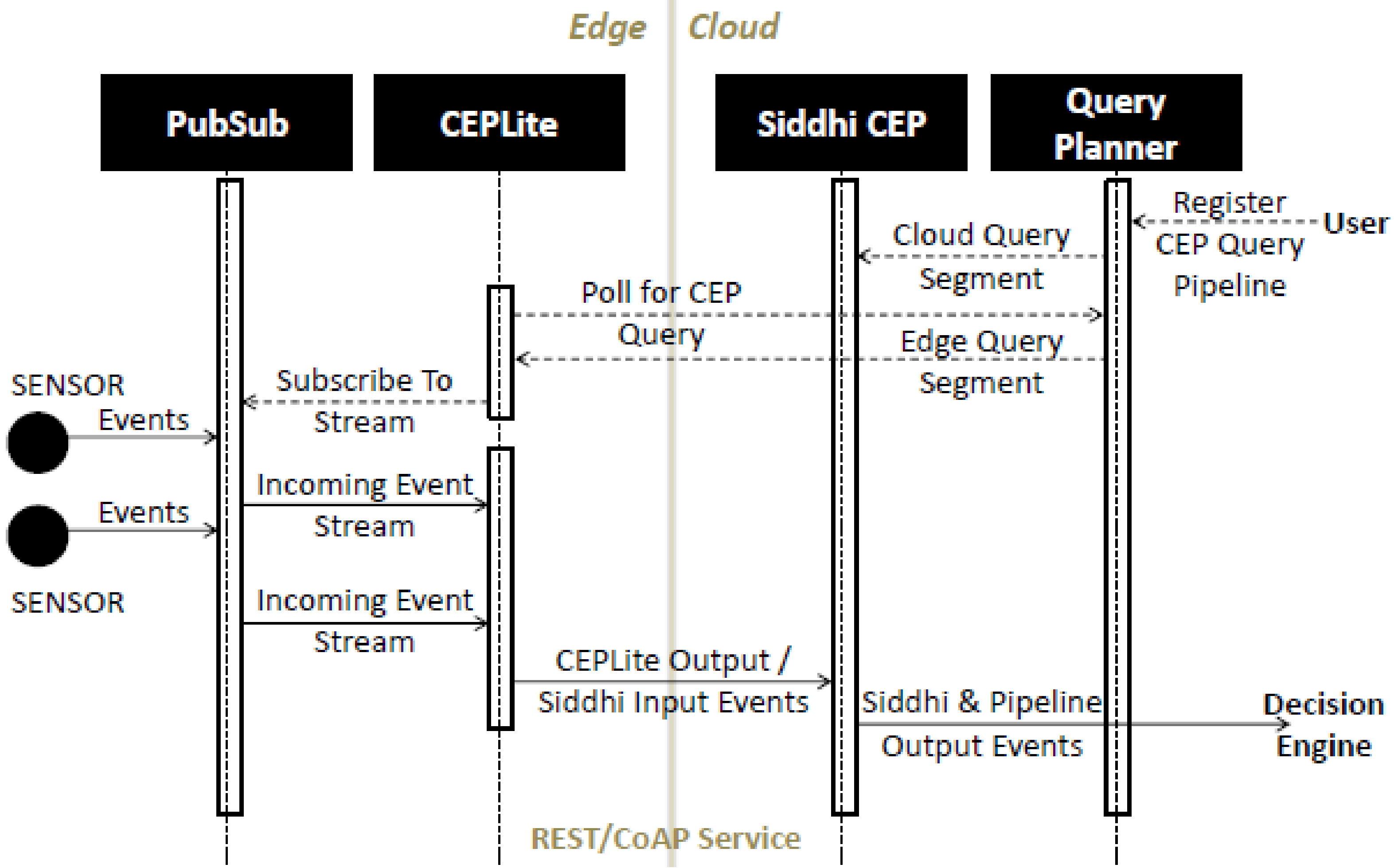
- Traditional CEP Processing has been centralized
  - But **IoT** Event sources are ***distributed***

CEP only on Cloud?	CEP only on Edge?
Latency, Privacy of moving Data from the Edge	Limited Expressivity & Compute Capability
Longer time to Respond	Need to integrate realtime with offline Big Data

- CEP in a distributed IoT environment
  - Capable edge devices, Smart Phones
  - Heterogeneous computing: Cloud + Edge
  - Distributed realtime analytics for IoT

Can we process event streams across Cloud & edge through efficient query partitioning to meet QoS Goals?

Edge+Cloud Sequence of Ops



Event Processing across Edge and the Cloud for Internet of Things Applications, Nithyashri Govindarajan, Yogesh Simmhan, Nitin Jamadagni and Prasant Misra, *COMAD*, Poster, 2014

Optimization Problem

- *Match a query* **Q** *within time* **T** *of input events*
- Constraints
  - Network latency
  - Data privacy
  - Compute capability
  - Expressivity
- Objective to Minimize
  - Execution co$ts
  - Energy consumption

Solution Approach

- Solve optimization problem using *dynamic programming*
  - Model query as a DAG.
  - Decide edge cut that meets objectives.
- Distributed CEP on Android & Cloud*
  - *CEPLite* engine on Android
  - Full featured *Siddhi CEP* on Cloud
- Deployment on IISc for Smart Campus
  - Sustainable water management
  - *Tank Overflow, Refill, Leakages, Quality*

Summary

Summary

- Clouds are *de facto* computing platforms
  - "Cloud first" & "Mobile first" for most new applications
- Big Data platforms are developed for Clouds & Commodity Clusters
  - Similar, but distinct distributed systems
- Clouds offer unique research challenges
  - Elasticity, cost, power, privacy, latency

Questions?

simmhan@serc.iisc.in

Acknowledgements

- Students & Collaborators
  - Ravikant Dindokar, DREAM:Lab, IISc
  - Neel Choudhury, DREAM:Lab, IISc
  - Viktor Prasanna, USC
  - Malati Hegde, ECE, IISc
  - Bharadwaj Amrutur, ECE, IISc
  - MS Mohankumar, Civil Engg, IISc
  - Rajesh Sundaresan, ECE, IISc

- Funding Agencies
  - DeitY, Government of India
  - Robert Bosch Centre for Cyber Physical Systems, IISc
  - Amazon AWS for Research
  - Microsoft Azure for Research
  - NetApp Inc.