Mobile App Acceleration via Fine-Grain Offloading to the Cloud

Chit-Kwan Lin

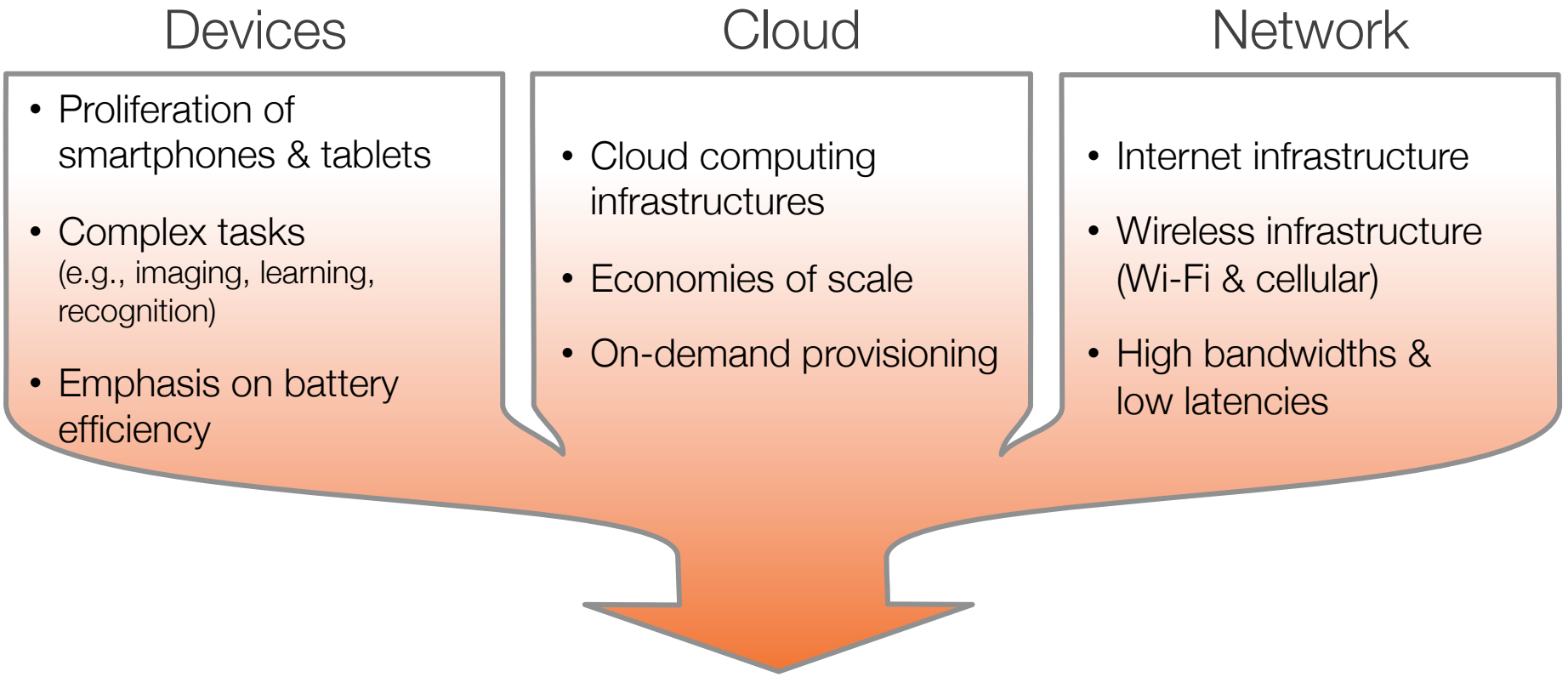
UpShift Labs

H. T. Kung

HARVARD
School of Engineering
and Applied Sciences

Copyright © 2014 UpShift Labs, Inc.

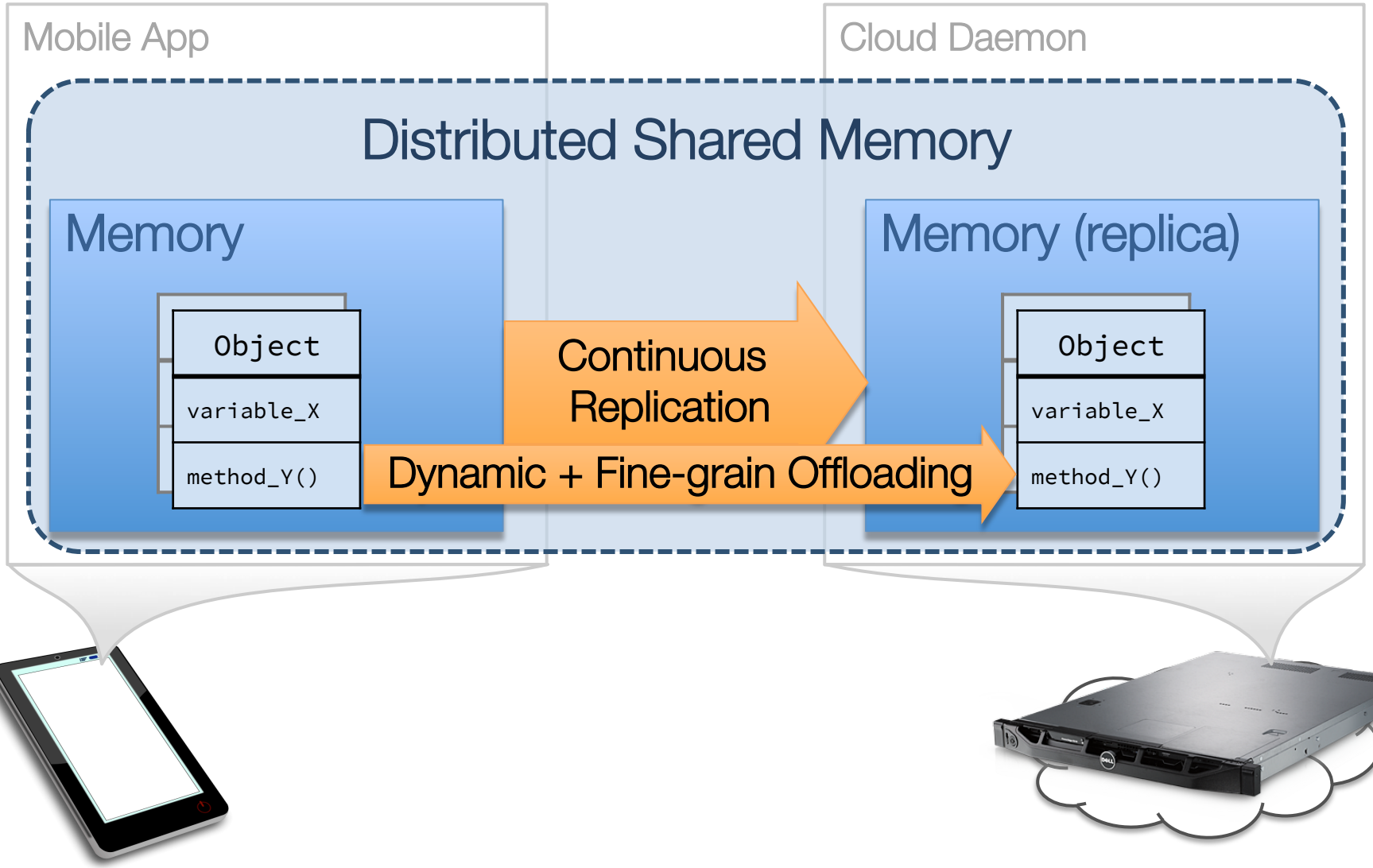
Confluence of Forces Points to Offloading to the Cloud



Compute Offloading

Copyright © 2014 UpShift Labs, Inc.

A Simple DSM Supports Offloading



* See last slide for image source references

Copyright © 2014 UpShift Labs, Inc.

Advantages of Dynamic + Fine-Grain

- **Dynamic**
  - Can offload arbitrary work at runtime
  - Can optimize resource utilization (e.g., battery) at runtime
- **Fine-grain**
  - At the level of a method invocation
  - Feels more responsive when failure requires local restart to fix
  - New class of small workloads for cloud providers
    - Useful for leveling out utilization while providing low-latency services for end-users

Copyright © 2014 UpShift Labs, Inc.

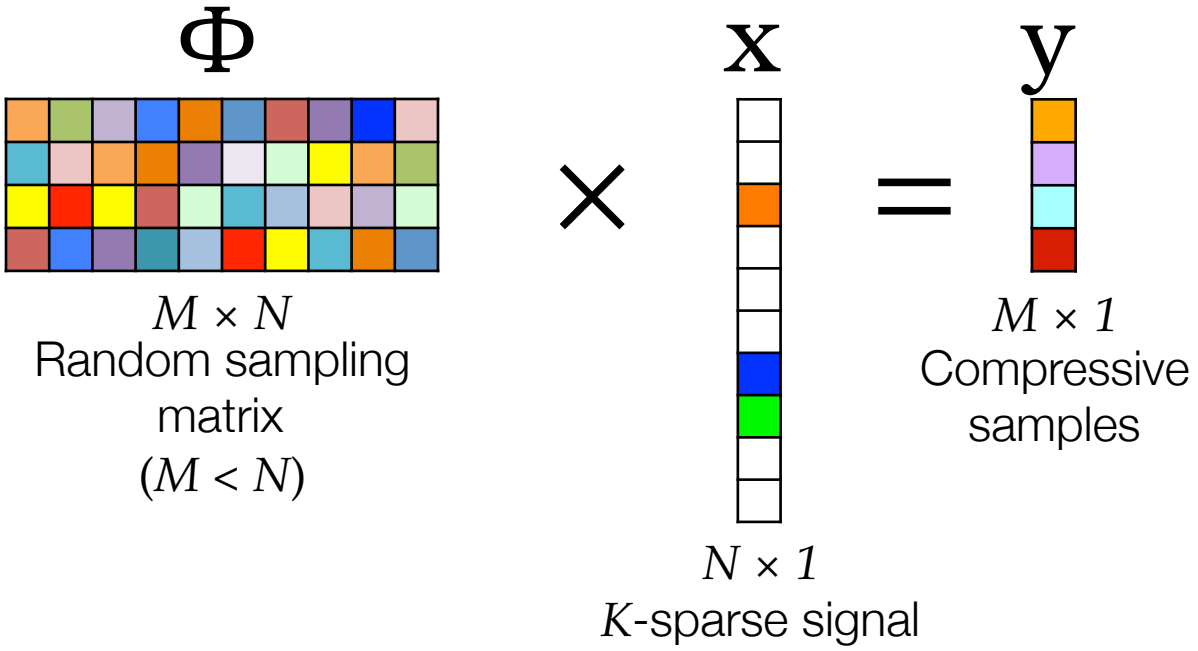
Challenges

- **Latency**
  - Dictates granularity: if update takes 5s, then workloads that run <5s don't benefit from offloading
- **Bandwidth**
  - We should only send deltas, but determining delta encoding has non-trivial costs
    - e.g., rsync can take 3 round trips (weak hash comparison, strong hash comparison, data) to generate a delta encoding, on top of time to calculate hashes
- **Compute**
  - We should compress to save bandwidth, but compression can be computationally expensive
- **Battery**
  - Shouldn't end up consuming more battery budget

Copyright © 2014 UpShift Labs, Inc.



Compressive Sensing



- Randomly mix signal elements by random projection onto lower-dimensional space
- Random Φ preserves Euclidean length/distance of sparse vectors with high probability when $M \geq \mathrm{O}(K \log N/K)$
- Decode $\mathbf{x}$ from $\mathbf{y}$ by solving $\mathbf{y} = \Phi\mathbf{x}$ via optimization (linear programming)

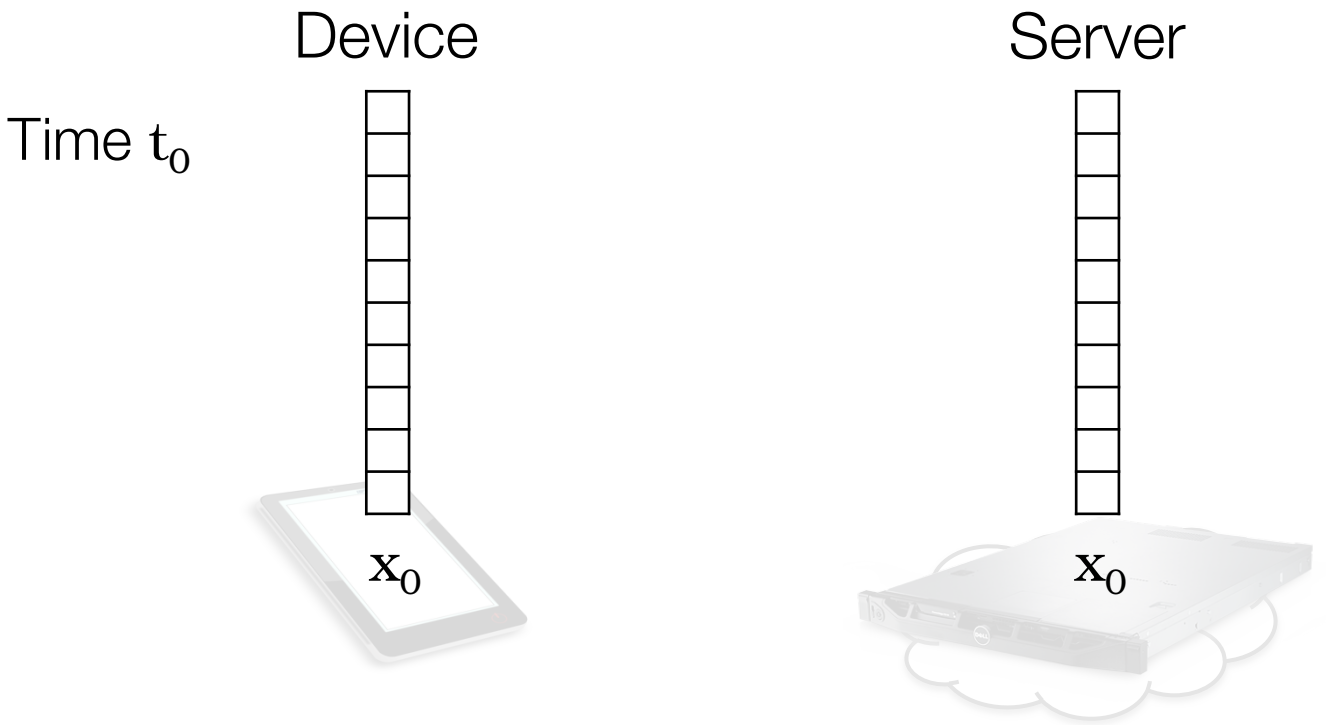
Key Insight



Writes (deltas) to memory typically constitute a sparse signal that can be compressively sampled

Copyright © 2014 UpShift Labs, Inc.

Compressive Replication (1)



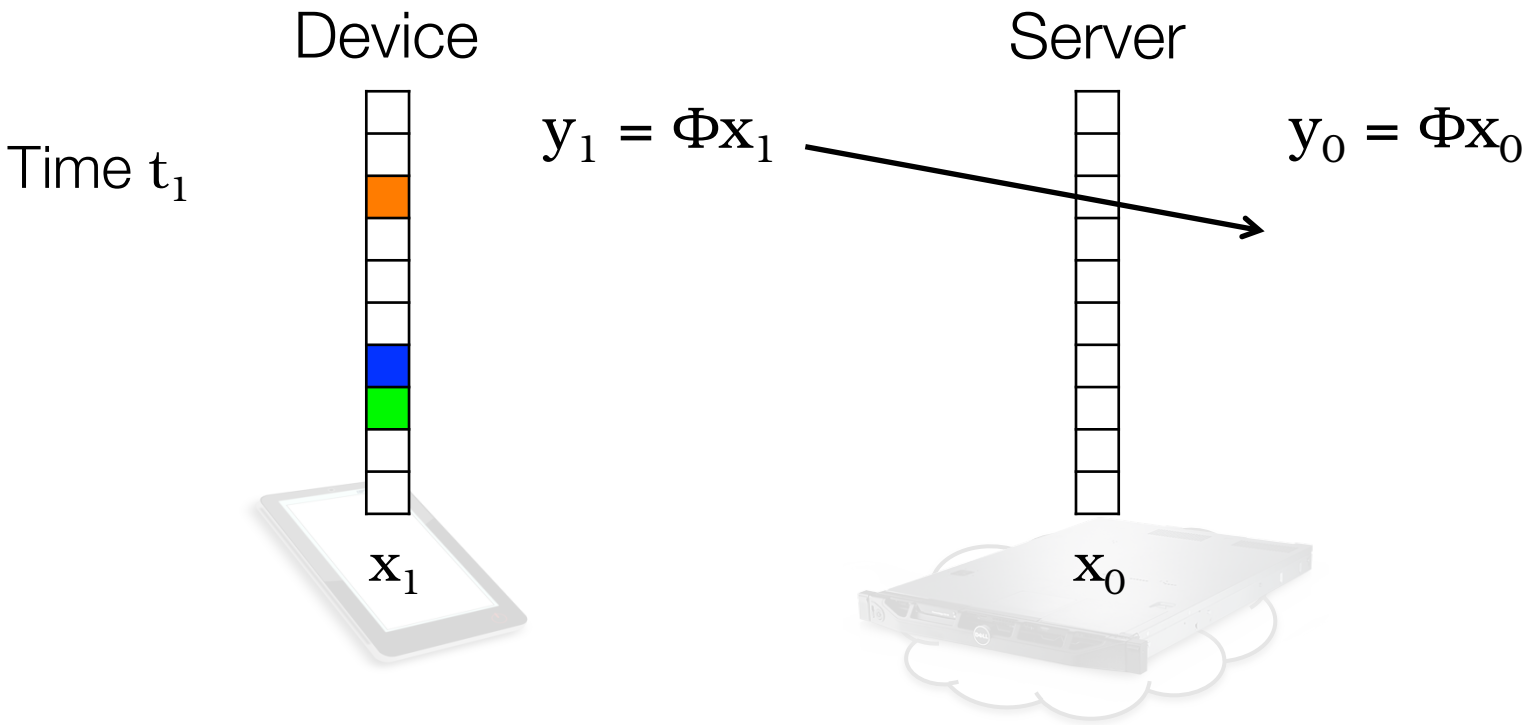
Memory starts out synchronized, with byte values $\mathbf{x}_0$

Both device and server know Φ

Server calculates $\mathbf{y}_0 = \Phi\mathbf{x}_0$

Copyright © 2014 UpShift Labs, Inc.

Compressive Replication (2)



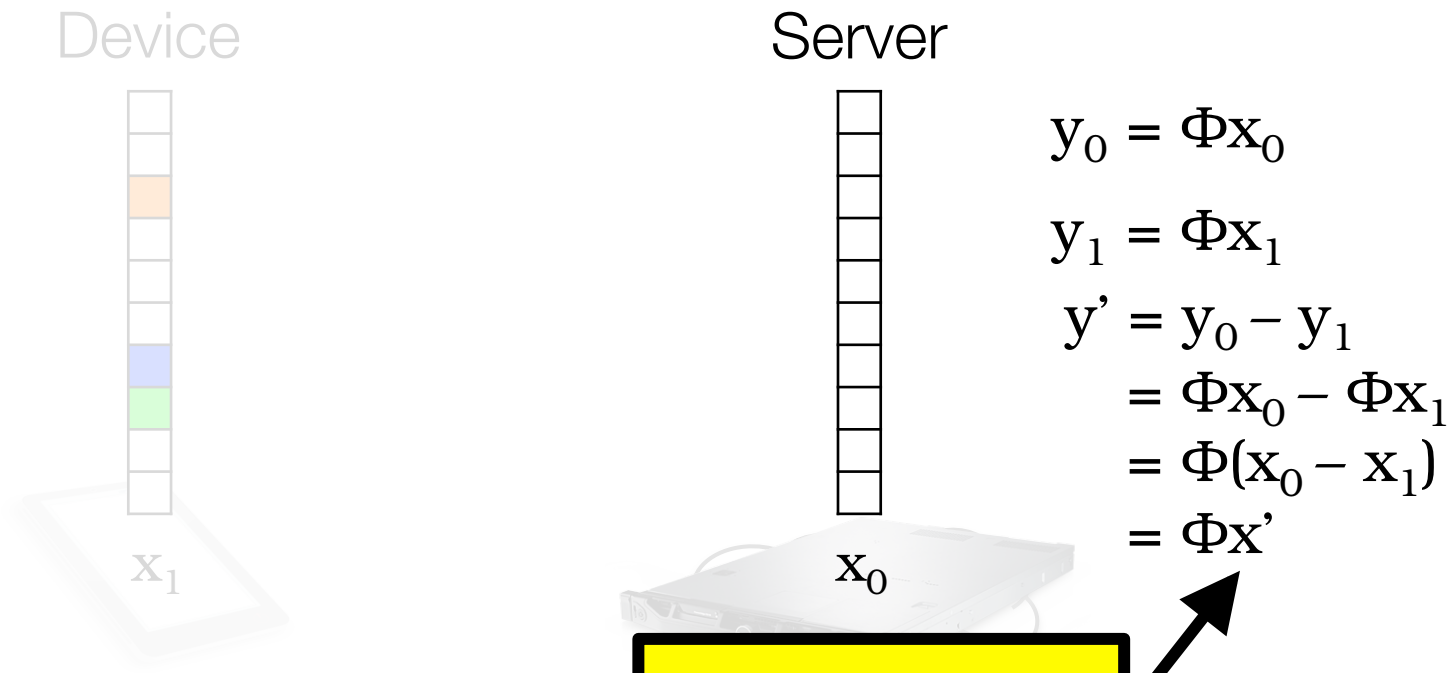
Some values in memory change, resulting in x_1

Device calculates $y_1 = \Phi x_1$

Device transmits $y_1 = \Phi x_1$ to server

Copyright © 2014 UpShift Labs, Inc.

Compressive Replication (3)



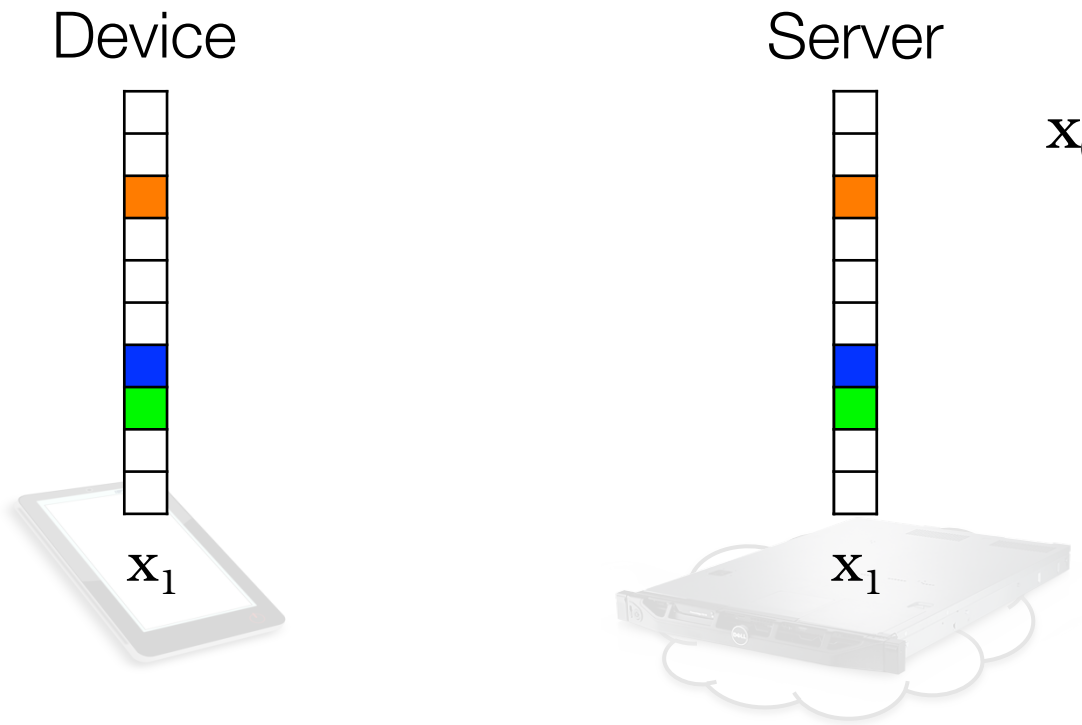
Server calculates $y' = y_0 - y_1$

$y' = \Phi x'$ has same form as compressive sensing decoding problem, so server solves for x'

Copyright © 2014 UpShift Labs, Inc.

Compressive Replication (4)

Time t_1



$$x_0 - x' = x_0 - (x_0 - x_1) = x_1$$

Server calculates $x_0 - x' = x_1$, which is the new memory state

Server is now updated

Copyright © 2014 UpShift Labs, Inc.

Novel Characteristics

- **All-in-one**
  - Delta encoding + compression
- **Delta encoding figured out by server, not device**
  - Automatically recovered during decoding
  - Just send compressive samples; no add'l network costs
- **Codec is resource-commensurate**
  - Device: low-complexity encoder
  - Server: higher-complexity decoder
  - Unlike traditional compressors

Copyright © 2014 UpShift Labs, Inc.

What do we compare against?

- **No similar replication methods**
  - Compressive replication gives all-in-one delta encoding + compression
  - e.g., rsync
    - Compression is an add'l step
    - Needs multiple round-trips to determine delta encoding

- **Compressed snapshots is a more appropriate comparison**
  - No add'l round-trip overheads
  - Just compress the whole memory page (snapshot)
  - snappy, zlib

- **Metrics**
  - Replicate 64KB memory block
  - Total Latency = encoding + network + decoding
  - Compression ratio (bandwidth cost)

Copyright © 2014 UpShift Labs, Inc.

Latency/Compression Trade-off

Near ~100ms threshold of user-perceptible app delay

	Encoding (ms)	Network (ms)	Decoding (ms)	**Total Latency (ms)**	**Compression Ratio**	Update Size (KB)
snappy	4	15	--	19	3.8 : 1	17.2
zlib	487	13	--	500	6.0 : 1	10.9
compressive replication	53	12	70	135	7.3 : 1	9.0

Highest compression ratio

Copyright © 2014 UpShift Labs, Inc.

Latency/Compression Trade-off

Remains good w/ high Kolmogorov complexity, which would confound snappy/zlib

	Encoding (ms)	Network (ms)	Decoding (ms)	**Total Latency (ms)**	**Compression Ratio**	Update Size (KB)
snappy	4	15	--	19	3.8 : 1	17.2
zlib	487	13	--	500	6.0 : 1	10.9
compressive replication	53	12	70	135	7.3 : 1	9.0

Unlike snappy/zlib, comp ratio fixed & b/w cost predictable because $y = \Phi x$ is independent of input's Kolmogorov complexity

Copyright © 2014 UpShift Labs, Inc.

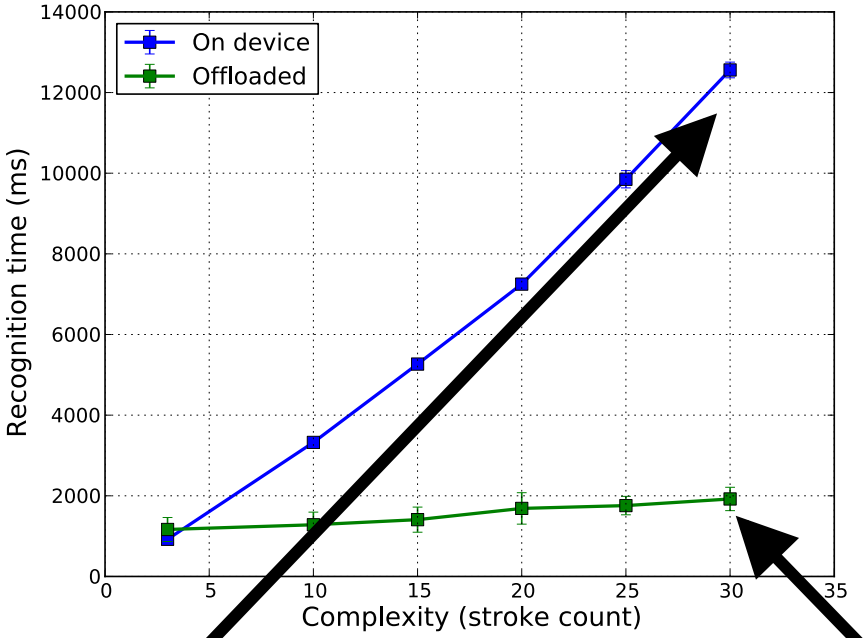
Example Handwriting Recognition App

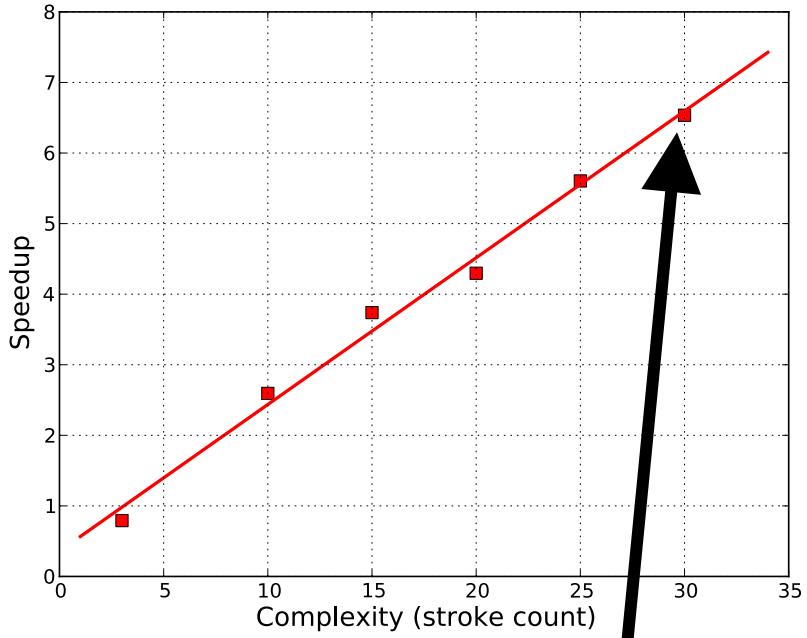
- UpShift platform for **compressive offloading** of iOS apps
- SVM for Chinese handwriting recognition
  - Stroke count is measure of task complexity
- Device: iPad (3rd gen)
- Server: Amazon g2.xlarge
  - GPU for compressive replication decoding
  - CPU for SVM evaluation
- Network: 802.11g
  - Office setting
  - 19ms RTT to us-east-1a server



Copyright © 2014 UpShift Labs, Inc.

App Speedup





On-device execution time scales poorly with task complexity

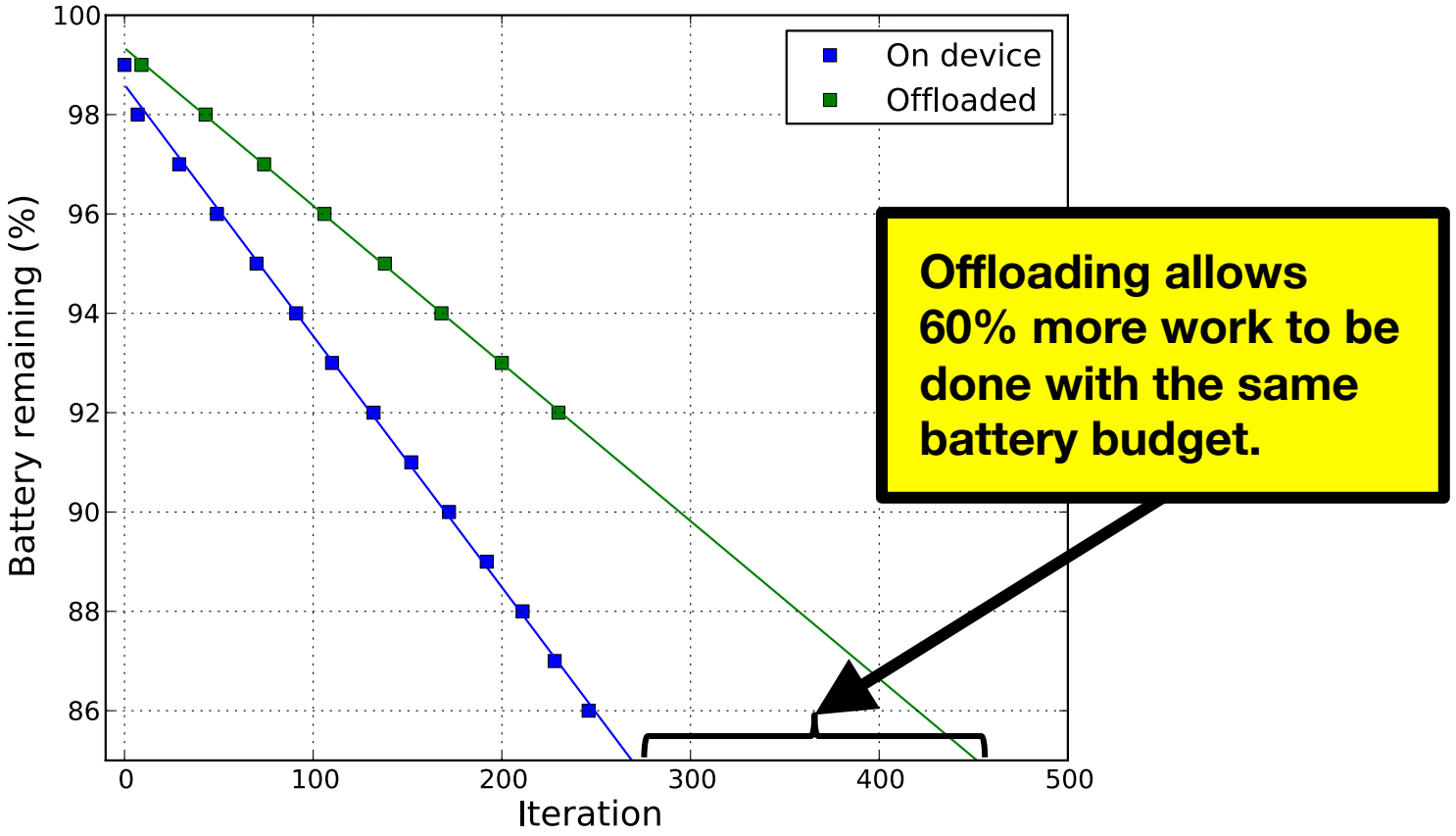
Offloaded execution time stays short due to low overhead of compressive offloading. Users expect this.

More complex tasks have greater speedup

Copyright © 2014 UpShift Labs, Inc.

Battery Savings

High task complexity (25 strokes), 250 iterations



Copyright © 2014 UpShift Labs, Inc.

Summary

- Low-latency DSM updates enable fine-grain offloading
- UpShift supports offloading ~100ms workloads, while keeping resource utilization low
- Achieves significant speedups and battery savings
- Key insight: memory writes are a sparse signal that can be compressively sampled
- Implications for future ARM-based DC's or x86-based devices

Copyright © 2014 UpShift Labs, Inc.

Questions?

ck@upshiftlabs.com

Copyright © 2014 UpShift Labs, Inc.