MobiSys 2008/TMC 2010/PerCom 2010/
MobiSys 2012/PerCom 2012/CACM 2012/
SenSys 2012

A mobile platform for context monitoring for PAN-scale dynamic mobile computing environments

Youngki Lee, Younghyun ju, Chulhong Min, Jihyun Yoo,Taewoo Park, Seungwoo Kang, Yoonseok Rhee, **Junehwa Song**

Dept. of Computer Science
KAIST

모바일 플랫폼의 진화

한국 모바일 산업 위기

한국 모바일 산업 위기?

피쳐폰	스마트폰 iPhone, Android	차세대 모바일 플랫폼

하드웨어 중심 통신 기능 집중	새로운 사용자 인터페이스 감성적 만족 추구	???

Or Nike+iPod?

By the way, what has happened in Korea about embedded systems?
why?

First Attempt: Supporting Proactive Applications

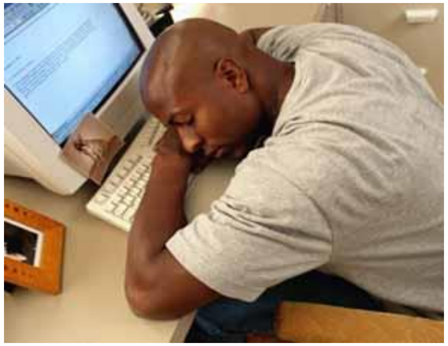
What are the requirements for true mobile App?

- Interface and interaction
  - Current status: touch, a bit of sound, a bit of motion
  - What would be good ?

- Proactiveness and personalization
  - Smartphone understands me better than my wife!
  - what is required for proactive services?
  - Example:

Example

Context-aware alarm

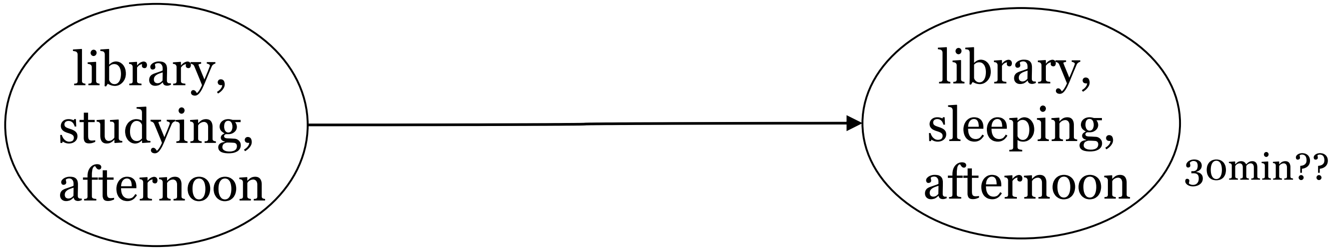


library, afternoon,
sleeping

30min??

Wake up!!

- Identify a change in the user context
  - Essential for Personal Context-aware Applications



Cf. Context reporting

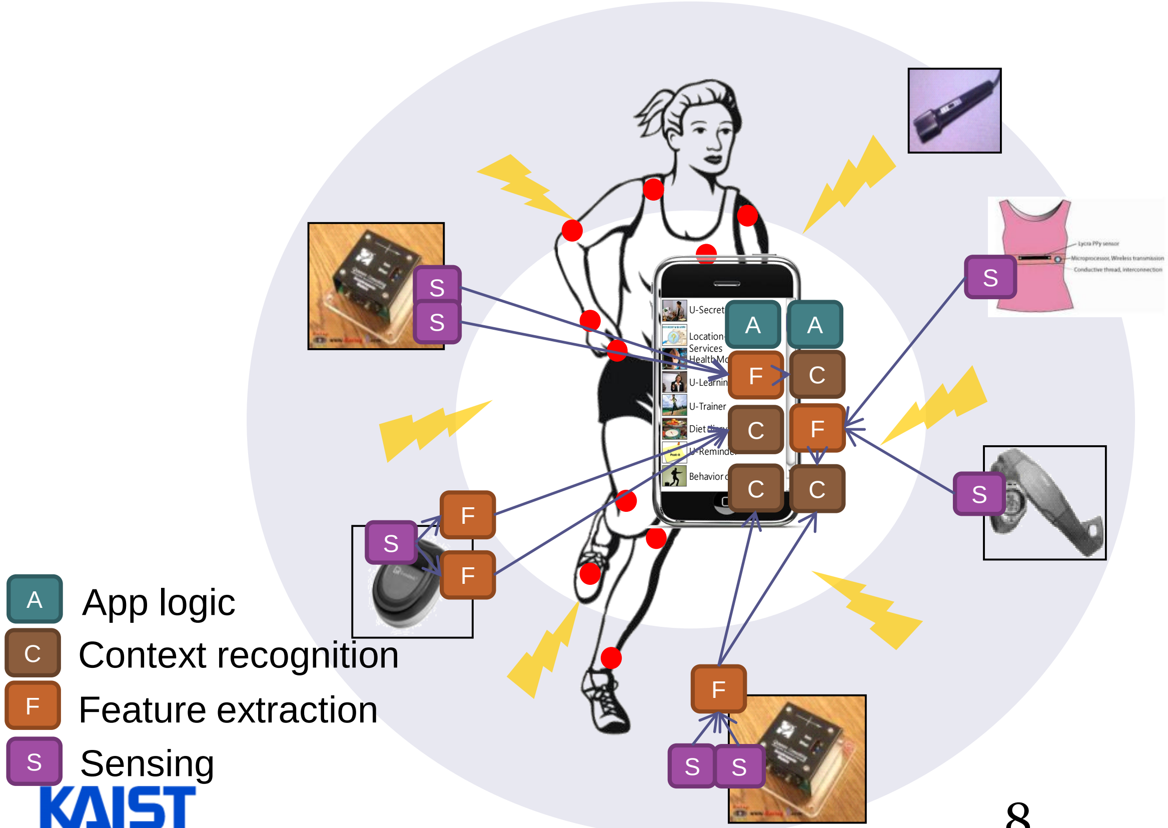
- Periodically report what the current user context is
- Report the user context whenever it satisfies a certain condition

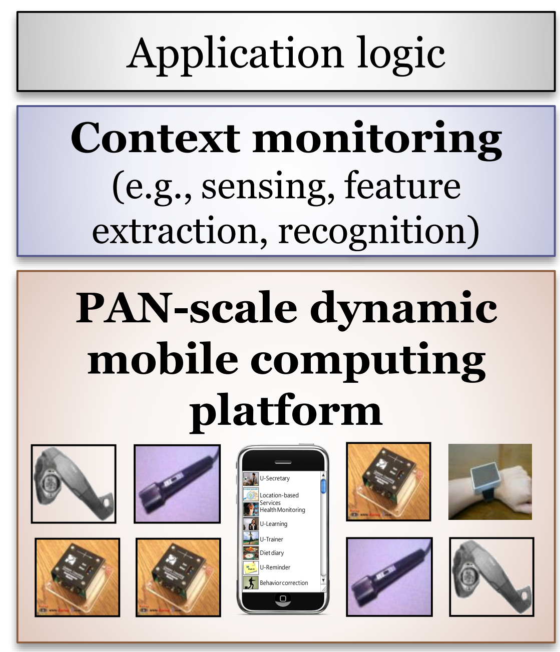
PAN-scale dynamic mobile computing

- A personal mobile device dynamically connected with many embedded, wearable, device-embedded sensors
- Serve a number of personal context-aware applications
  - providing proactive, personalized, situation-aware services

Continuous context monitoring

- Continuous monitoring of users' context
  - A key building block for personal context-aware applications
- Often requires complex, multi-step, continuous processing
  - Over multiple devices
  - E.g. Running situation : sensing in three 3-axis accelerometers, FFT processing, recognition





Challenges and Approach

Challenges

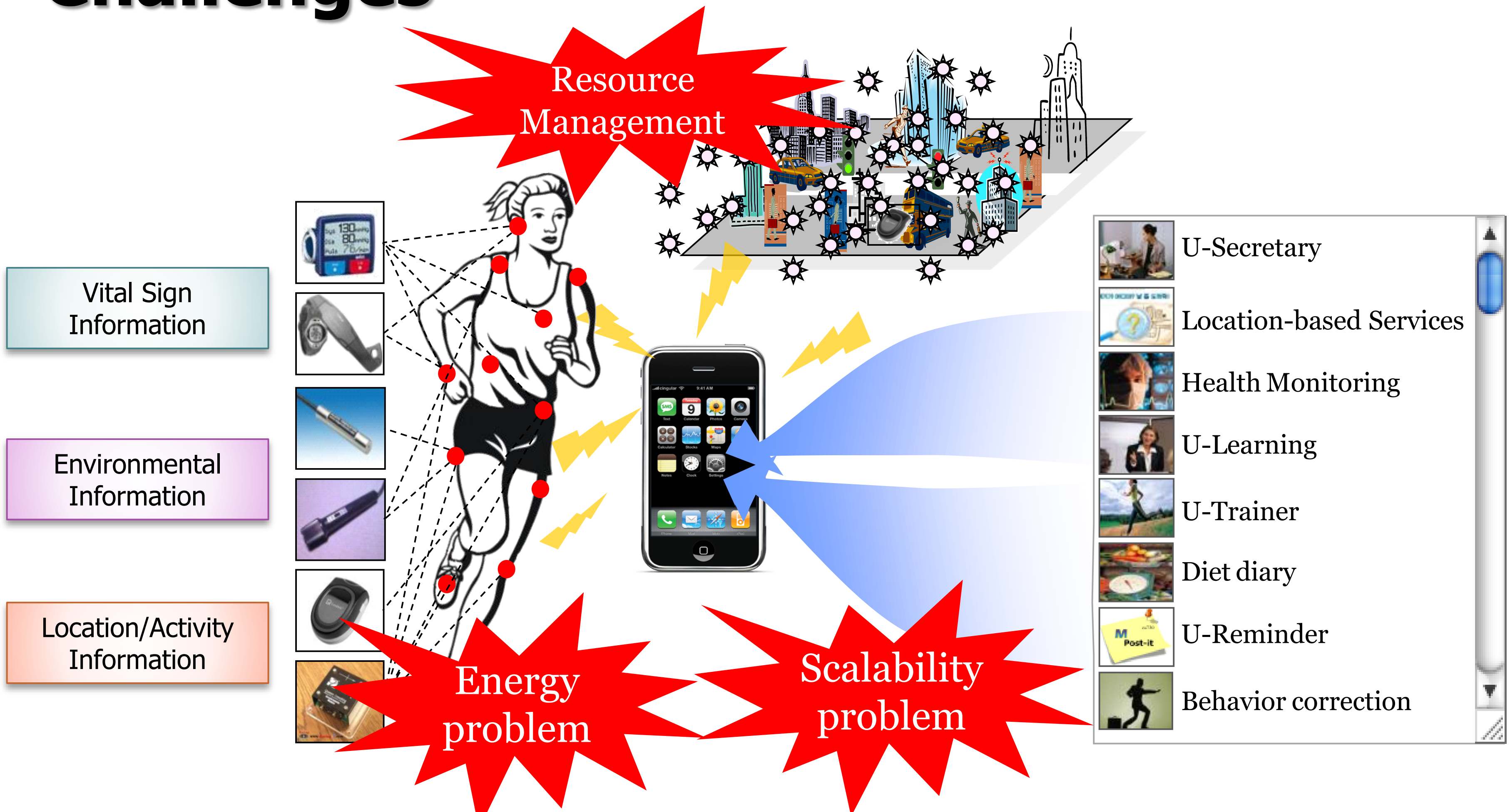
- A number of applications share highly scarce resources of the PAN-scale dynamic mobile computing environment
  - Only a few applications can run due to resource constraint
  - Potential capacity drop due to skewed resource utilization

- Significantly scarce resources
  - E.g., MicaZ Motes: 8MHz CPU, 4KB RAM, ~50Kbps Bandwidth
    - A light FFT library, *kiss_fft*, requires 40KB RAM, 10 MHz CPU
  - Limited battery power due to mobility

- Dynamic join/leave of heterogeneous sensors
  - E.g., take off a watch sensor, enter a smart space with sharable environmental sensors

- Dynamic changes in resource demands and status
  - Applications join and leave, registers and de-registers new and existing requests
  - Sudden drops in BW availability due to mobility, obstacles, …

Challenges



Tens/hundreds of heterogeneous sensors

Continuous data updates (Thousands of updates/sec)

Resource limitation (Battery, processor, memory)

Hundreds/thousands of long-running monitoring requests to detect context changes

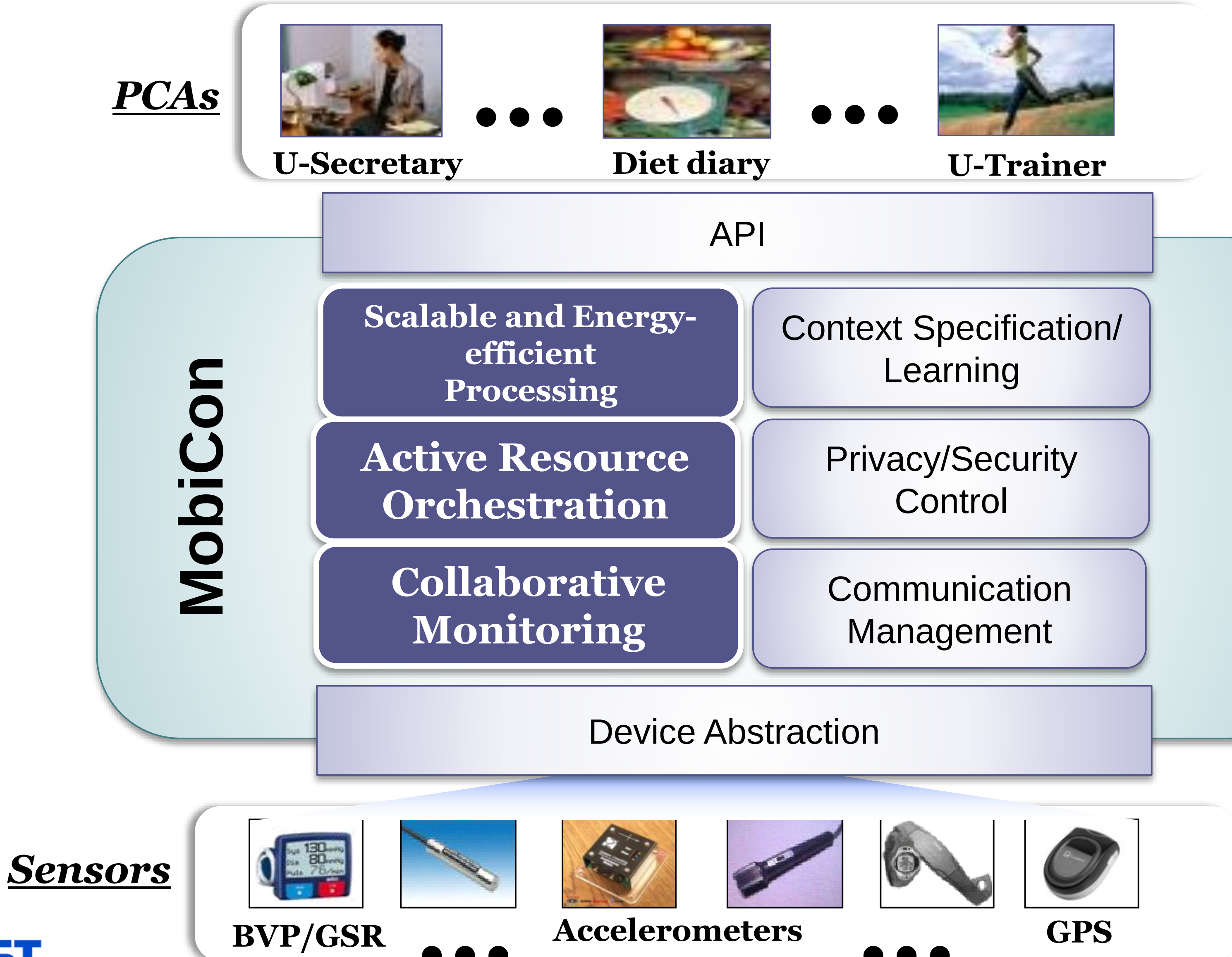
Applications themselves cannot solve the challenges

- Context monitoring is complexity thing to do!
  - Burdensome programming and debugging over sensor devices
  - Implementation of a range of processing modules for feature extraction and recognition
  - Repeated and time-consuming job for training (learning)

- It should handle (use and schedule) scarce resources over multiple devices
  - Awareness of the required amount of resources
  - Handling dynamic sensor availability and resource status
    - A variety of different resource conditions

- It should coordinate the resource use of other (multiple) applications
  - Should make applications communicate and negotiate with each other
  - However, it is almost impossible to make it

A new mobile platform for PAN-scale dynamic mobile computing environment



Current Status

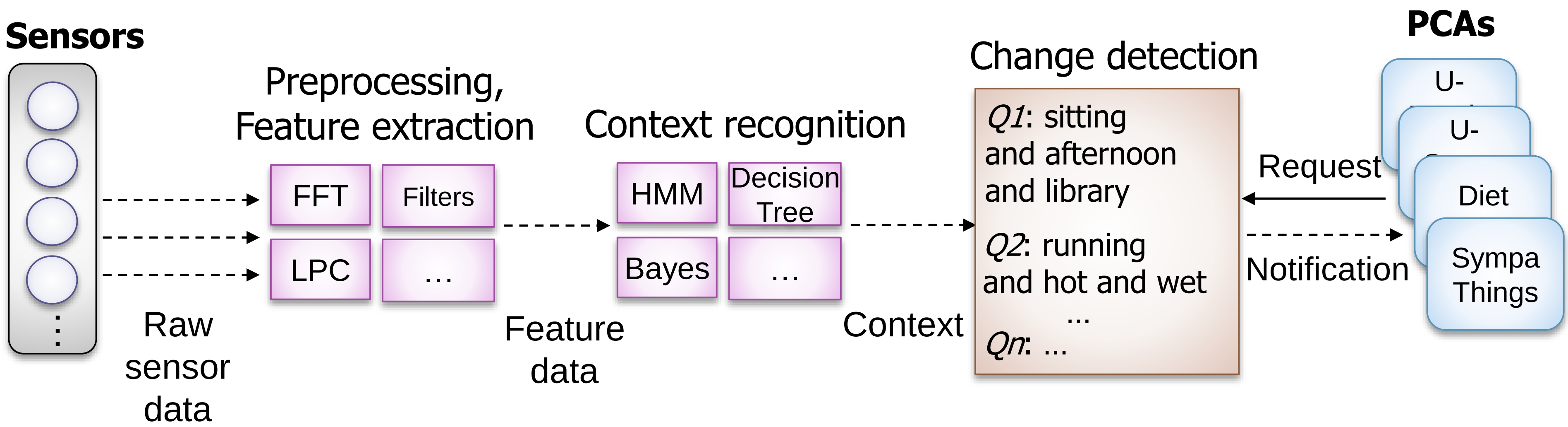
- Scalable and Energy-Efficient Context Processing: SeeMon (MobiSys 2008, TMC 2010)
- Active Resource Orchestration: ActraMon (PerCom 2010, TMC)
- Producer-oriented Execution of Sensing Flows: FastFlux (percom 2012)
- Orchestration of Multiple Sensing Applications on a Smartphone: SymPhoney (sensys 2012)
- Collaborative Context Monitoring: CoMon (MobiSys 2012)

Scalable and energy-efficient context monitoring

Computation & Energy Efficient Context Processing

- Bidirectional Processing Pipeline
- Transformation-based Approach
- Shared, Incremental Processing
- Producer-Oriented Context Processing
- Essential Sensor Set

General processing model for context/activity



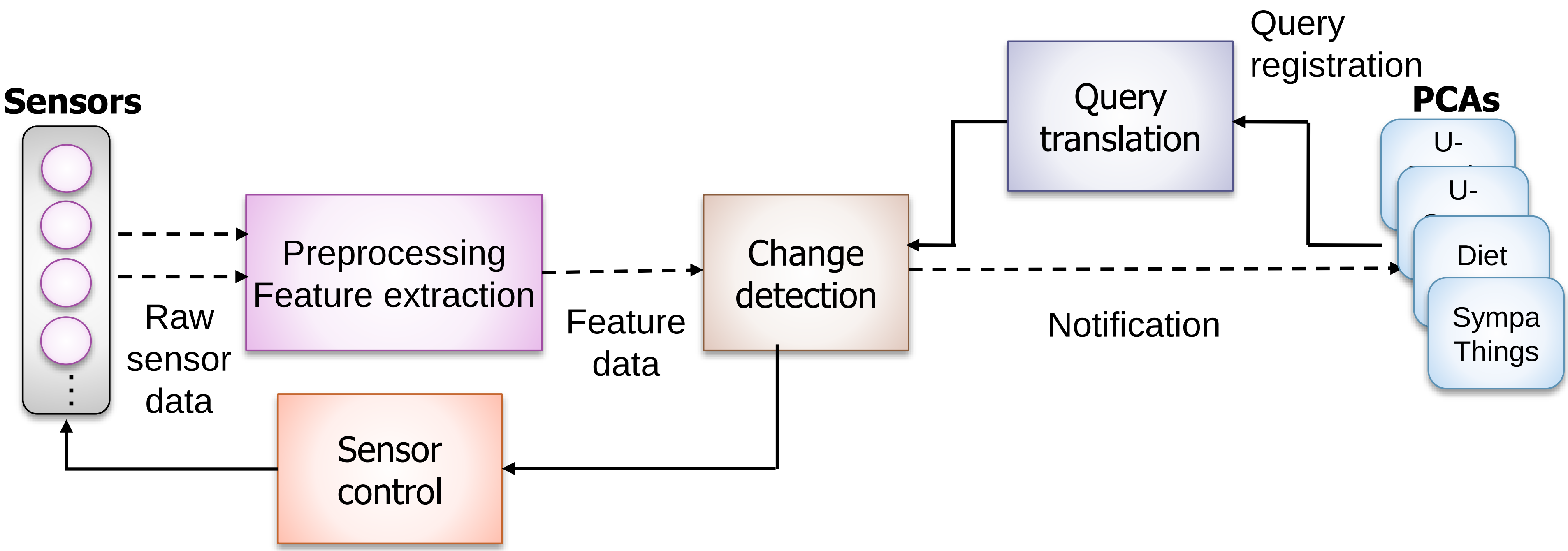
- Requires costly operations for
  - Continuous data updates from sensors
  - Continuous context processing
    - Complex feature extraction and context recognition
  - Continuous change detection
    - Repeated examination of numerous monitoring requests

KAIST

NC Lab Network Computing

Our approach: bidirectional processing

- Early detection of context changes
  - Remove processing cost for continuous context recognition
- Utilize the locality of feature data in change detection
  - Reduce processing cost by evaluating queries in an incremental manner
- Turn off unnecessary sensors for monitoring results
  - Reduce energy consumption for wireless data transmission



Context Monitoring Query (CMQ)

- Simple and intuitive query language
  - Free developers from the complexity of continuous context monitoring
  - Support the semantics to catch the context change

CONTEXT <context element>
(AND <context element>)*
ALARM <type>
DURATION <duration>

- Example query
  - *"Let me know if the user starts to run in a hot and humid weather"*

CONTEXT (activity == running) AND (temp == hot) AND (humidity == wet)
ALARM F → T
DURATION 1 month

Context elements

19

CMQ translation

- Avoid context recognition processing and enable feature data-level change detection

CONTEXT (activity == running) AND (temp == hot) AND (humidity == wet)
ALARM F → T
DURATION 1 month

Feature data-level CMQ

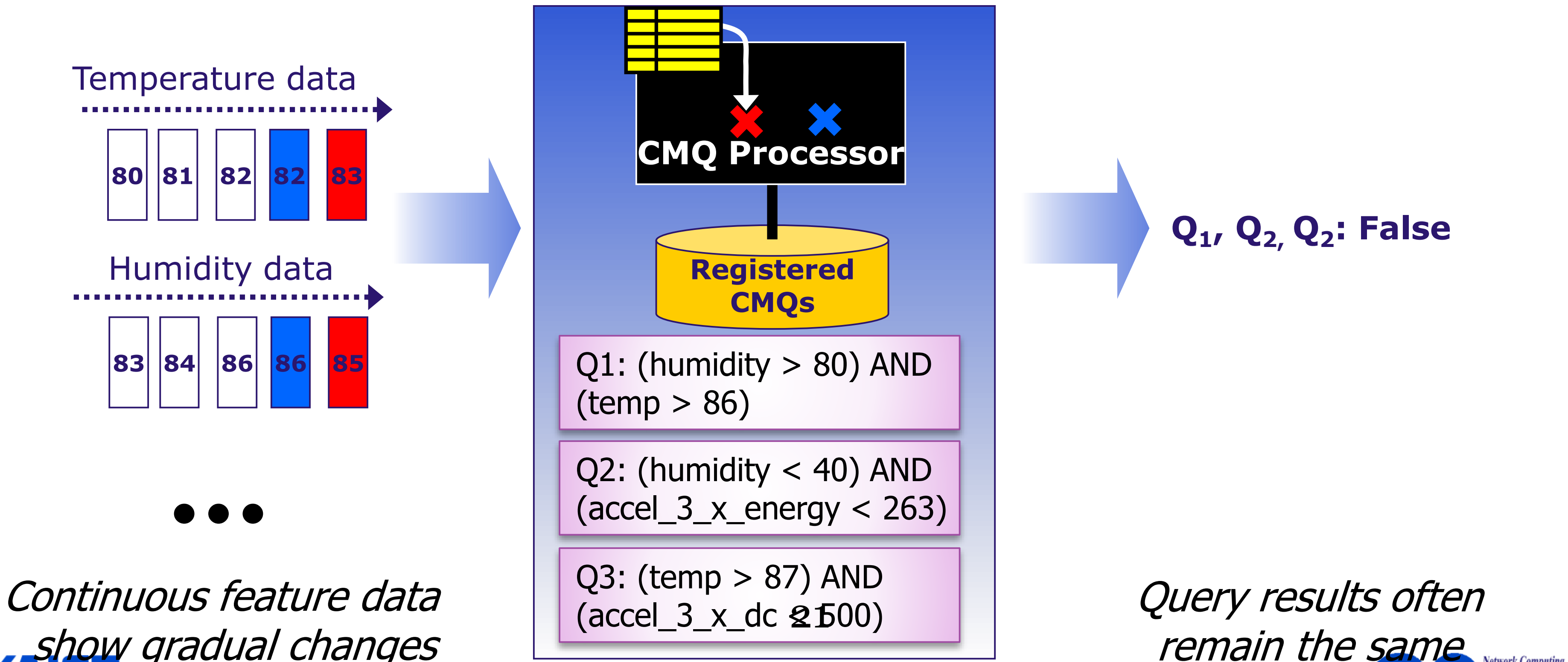
Context Translation Map

Context-level semantic										
Type	Value	Feature1			Feature2			Feature2		
		ID	Low	High	ID	Low	High	ID	Low	High
activity	running	accel_1_y_energy	52		accel_3_x_dc		500	accel_3_x_energy		263
temp	hot	temp	86 F							
humidity	wet	humidity	80%							
…	…									

```
CONTEXT (accel_1_y_energy > 52 ) AND (accel_3_x_dc < 500 )
  AND (accel_3_x_energy < 263 ) AND (temp > 86 ) AND
  (humidity > 80)
ALARM F → T
DURATION 1 month
```

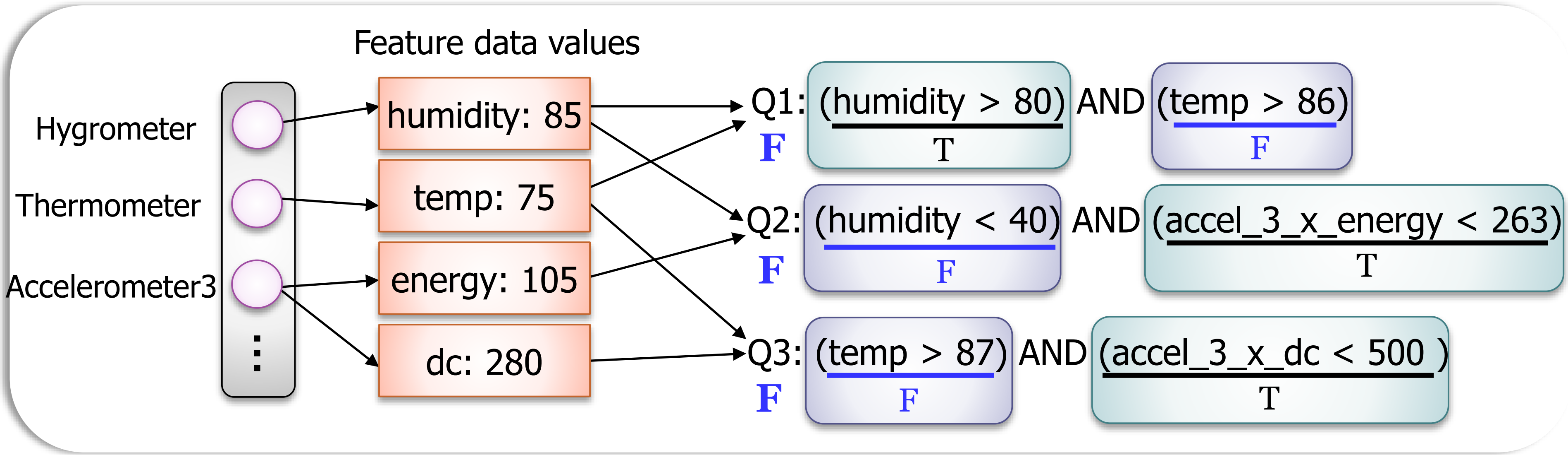
Shared and Incremental Context Processing

- Incremental processing by exploiting the locality of context
  - Locality of context ➔ locality of feature data
- Shared processing on all registered CMQs



Essential Sensor Set: Energy-efficient context monitoring

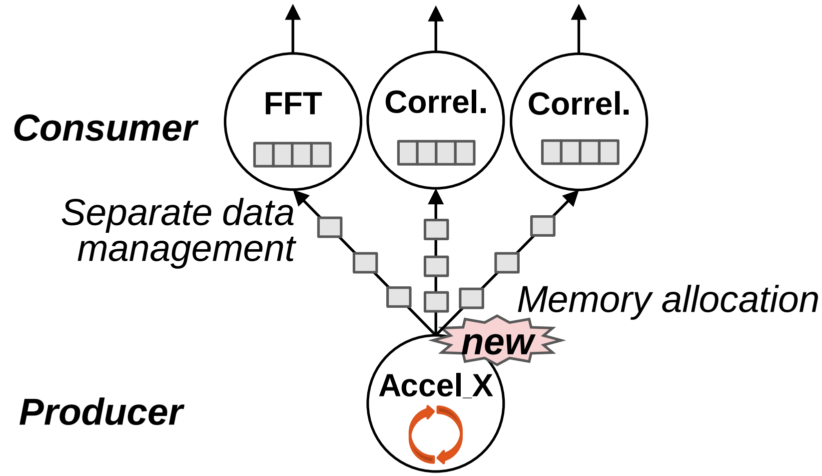
- Avoid unnecessary data transmission from wireless sensors
- Identify and deactivate unnecessary sensors
  - **Example query:** "*Is the weather hot and humid?*"
    - If it is already cool, no need to see humidity.

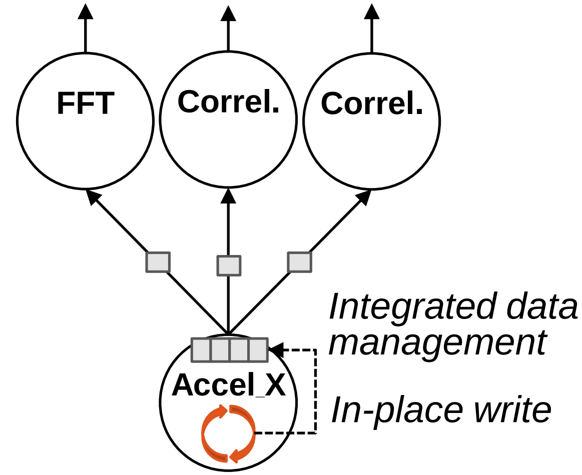


22

Fast-Flux: Producer-oriented Execution of Sensing Flows

Approach: Producer-oriented Execution





Consumer-oriented model:
Consumers collect, manage, and process their input data

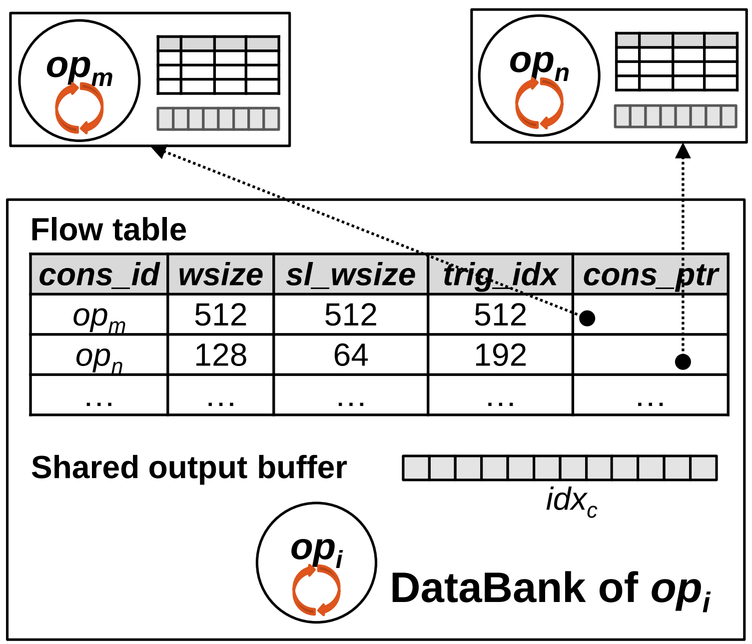
- Frequent messaging and scheduling overhead
- Redundant management of the same data
- Repetitive memory allocation and de-allocation

Producer-oriented model:
Producers manage their output data in an integrated fashion

- *Lazy data delivery*: reduces the number of data pass and operator scheduling operations
- *Integrated data management*: eliminates the redundancy in the separate management
- *In-place buffer write*: avoids repetitive memory allocation and de-allocation

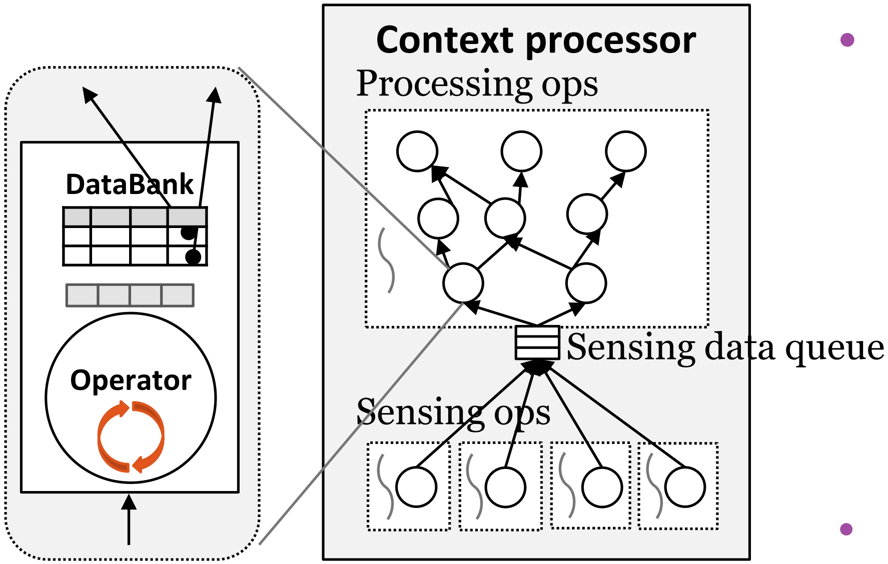
DataBank

- Execution container for an operator that realizes the producer-oriented model
  - Serves as the basic unit in the execution of a dataflow graph
  - Takes charge of the management and delivery of the output data
  - Separates the pure processing logic from the logic for dataflow execution and data management



cons_id	*wsize*	*sl_wsize*	*trig_idx*	*cons_ptr*
op_m	512	512	512	●
op_n	128	64	192	●
…	…	…	…	…

Execution Network



- A network of DataBanks for the execution of a dataflow graph
- DataBanks are connected via the pointers in internal flow tables
  - A DataBank passes data to next DataBanks through direct function call via the pointer
    - Low overhead in inter-operator communication
- DataBanks of sensing operators are connected to their consumers' DataBanks via sensing data queue
  - Due to the asynchronous nature of sensing operators

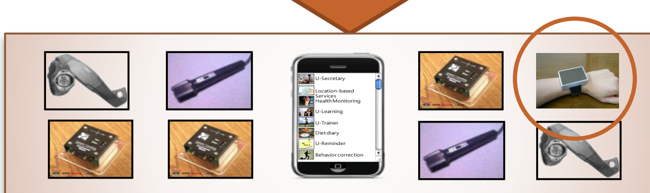
Active Resource Orchestration

Active resource use orchestration

active resource use orchestration vs. *passive resource use mngmnt*

(e.g., in conventional resource management systems in mobile systems, sensor systems)

Active resource use orchestration	Passive resource use mngmnt
✓ High-level context monitoring request E.g. Context == Running	✓ Low-level resource allocation request E.g. 5MHz CPU, 5KB RAM, 10kbps BW for a watch sensor
✓ System-wide holistic view of applications and resources	✓ Has limited view , i.e., the resource requests
✓ Flexible system-driven resource binding	✓ Static app.-driven resource binding

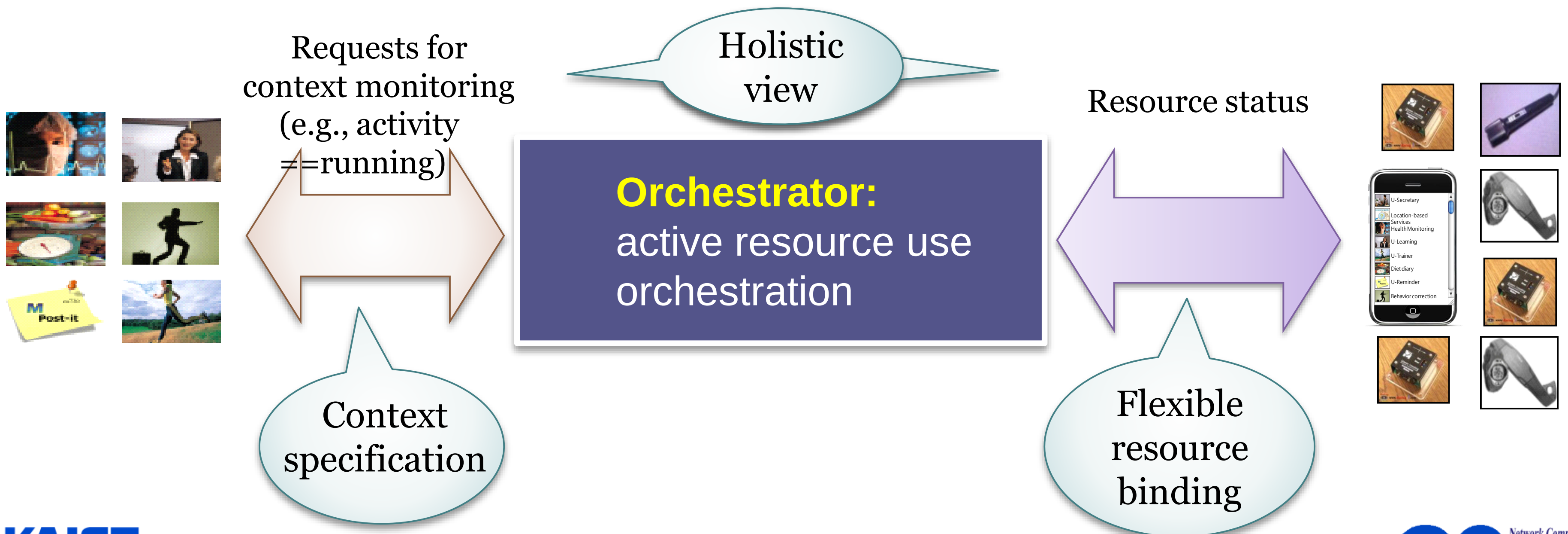


KAIST

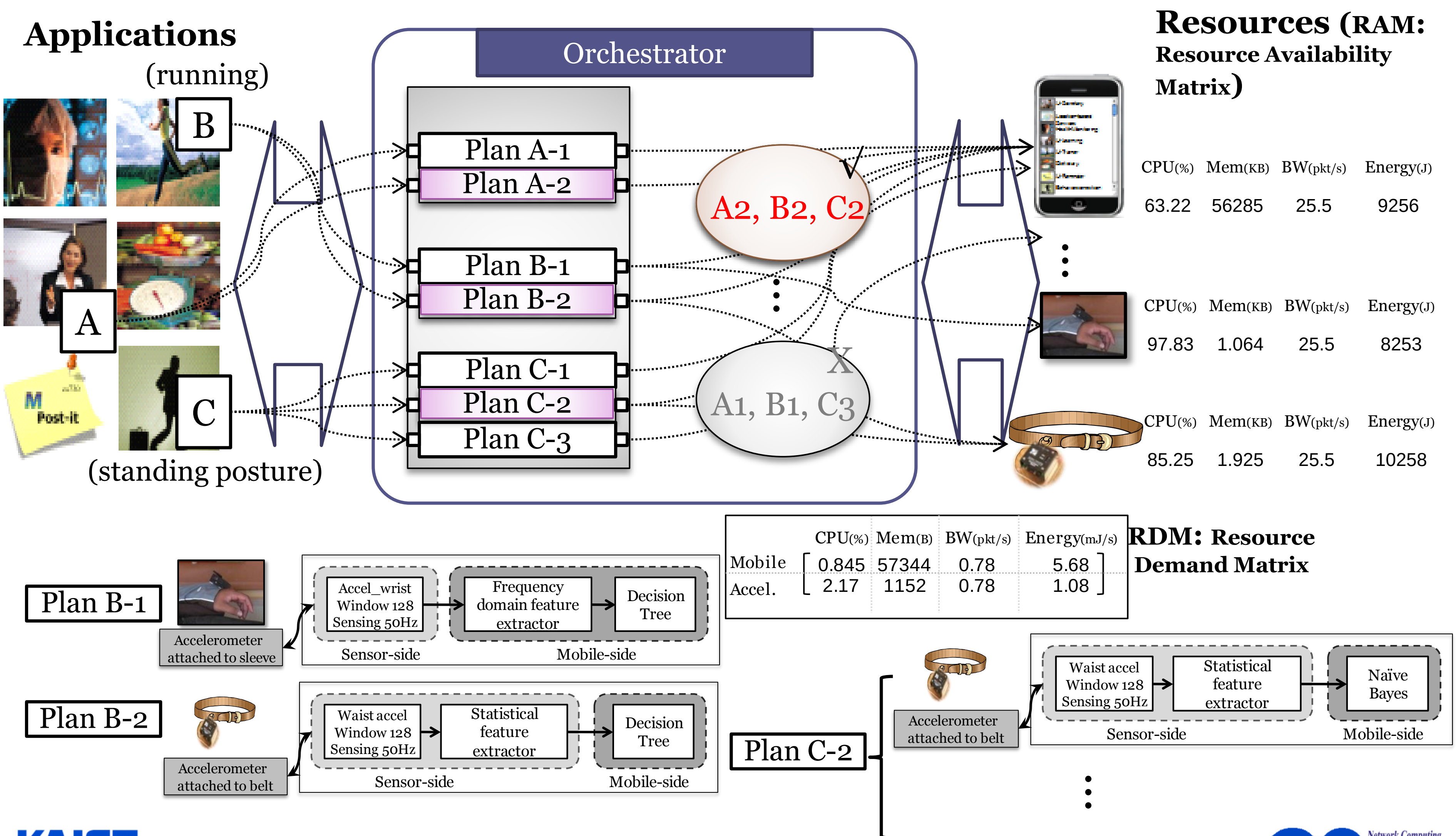
Network Computing lab

Key features

- Alternative resource usage plans
  - Context → a variety of processing methods → diverse resource usages (device, task distribution)
- Runtime decision of the best use (by system-wide policy)



Operation example



SymPhoney:

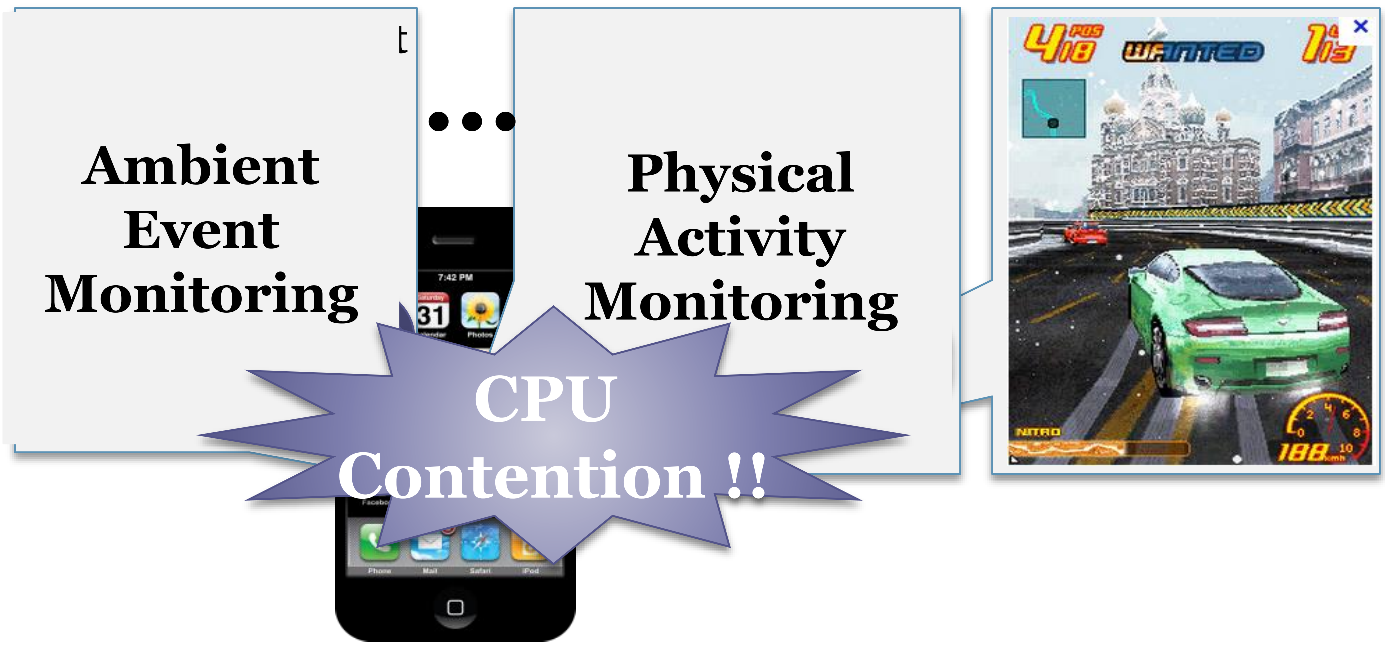
Orchestration of Multiple Sensing Applications on a Smartphone

KAIST

Network Computing Lab

Concurrent Sensing Applications

- CPU contention
  - **Concurrent** sensing applications
  - **Continuous and heavy** CPU consumption (e.g., 5% ~ 20%)
  - **Resource-limited** smartphone
  - Users won't want to use 100% of their CPU for BG jobs



CPU Contention Results in …

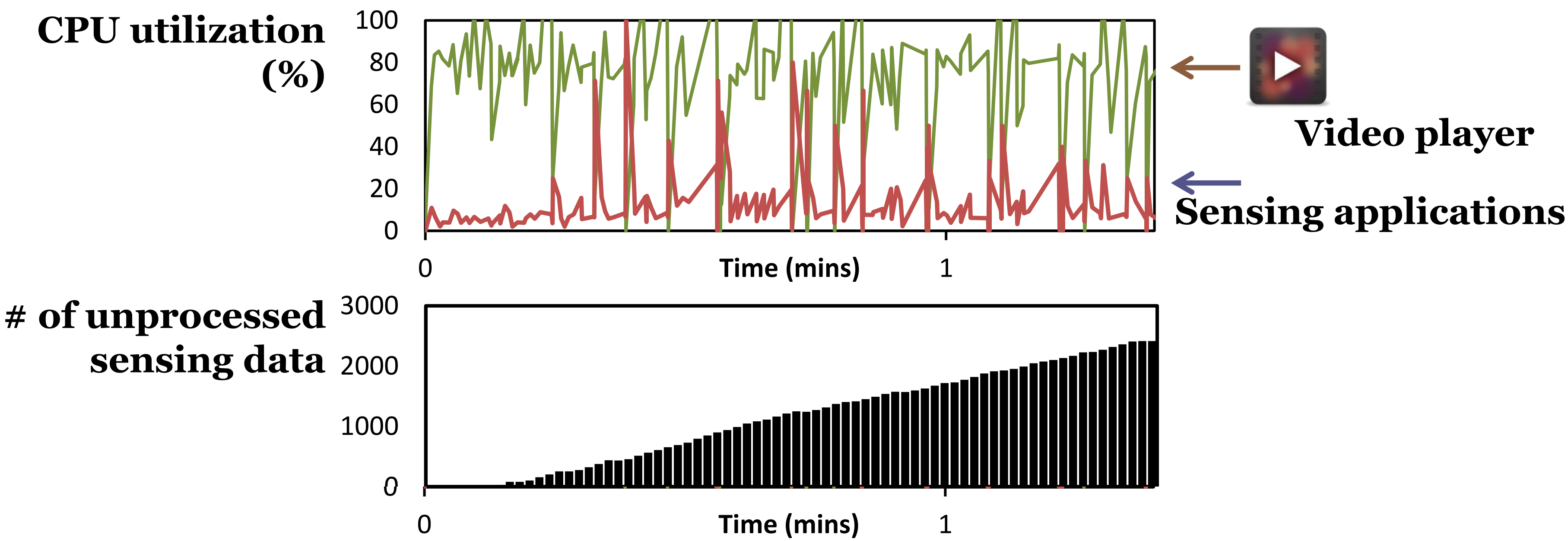
- Sensing apps can't process continuous sensing data on time !! → **processing delay and data drop**
  - Long service delay
  - Abrupt service vacancy
  - Wrong context inference,

  which are critical for timely, situation-aware services

- **Performance degradation of foreground apps**
  - e.g., frame drop in games, increase of web loading time

Current Mobile OS

- Allocate most of the CPU time to an activated FG app
  - e.g., more than 90% in Android
  - Multiple BG sensing apps should share the remaining small portion



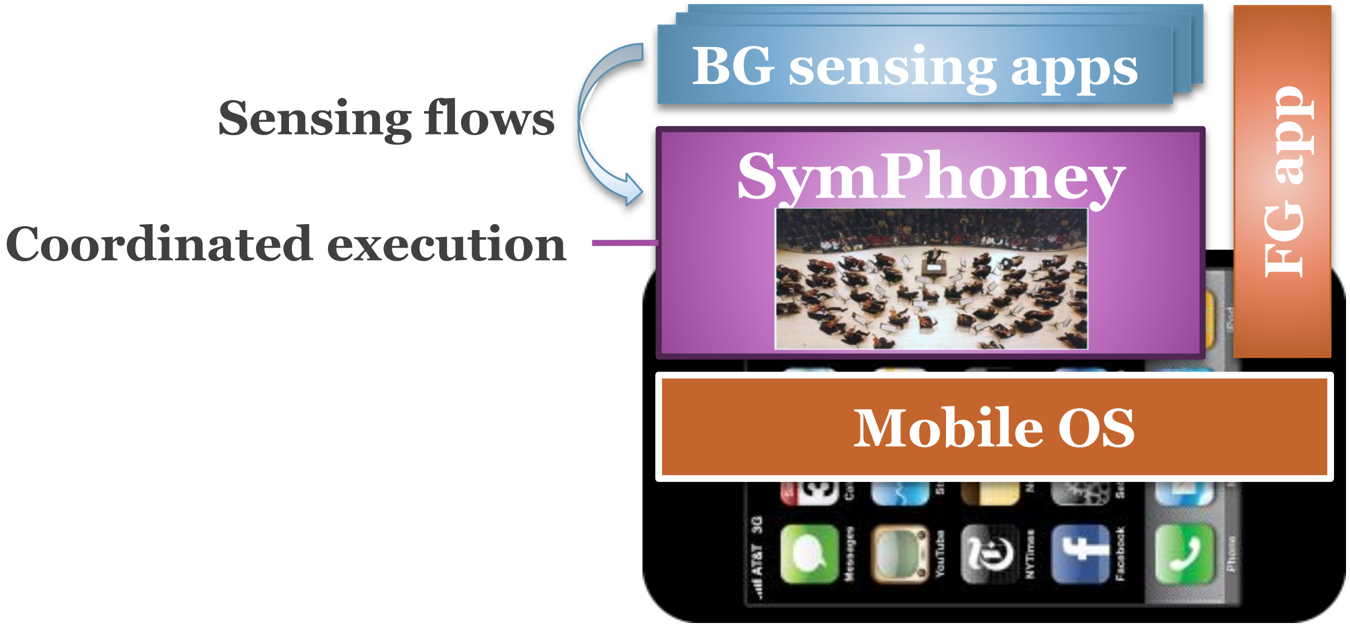
Current Mobile OS

- Hard to determine the proper amount of CPU time for each sensing app
  - Sensing apps are dealt with just as **black-box processes**
  - No application-level information such as
    - the CPU time required to process continuous sensing data
    - the required QoS level

→ **Could result in unfair and inefficient resource use**

SymPhoney

- A sensing flow execution engine for concurrent sensing apps
  - Coordinate the resource use of contending applications
    - Maximize their utilities under given resource conditions
  - Adapt to the fluctuating resource availability from FG apps
    - Minimize the performance degradation of the interactive apps



Flow-Aware Coordination

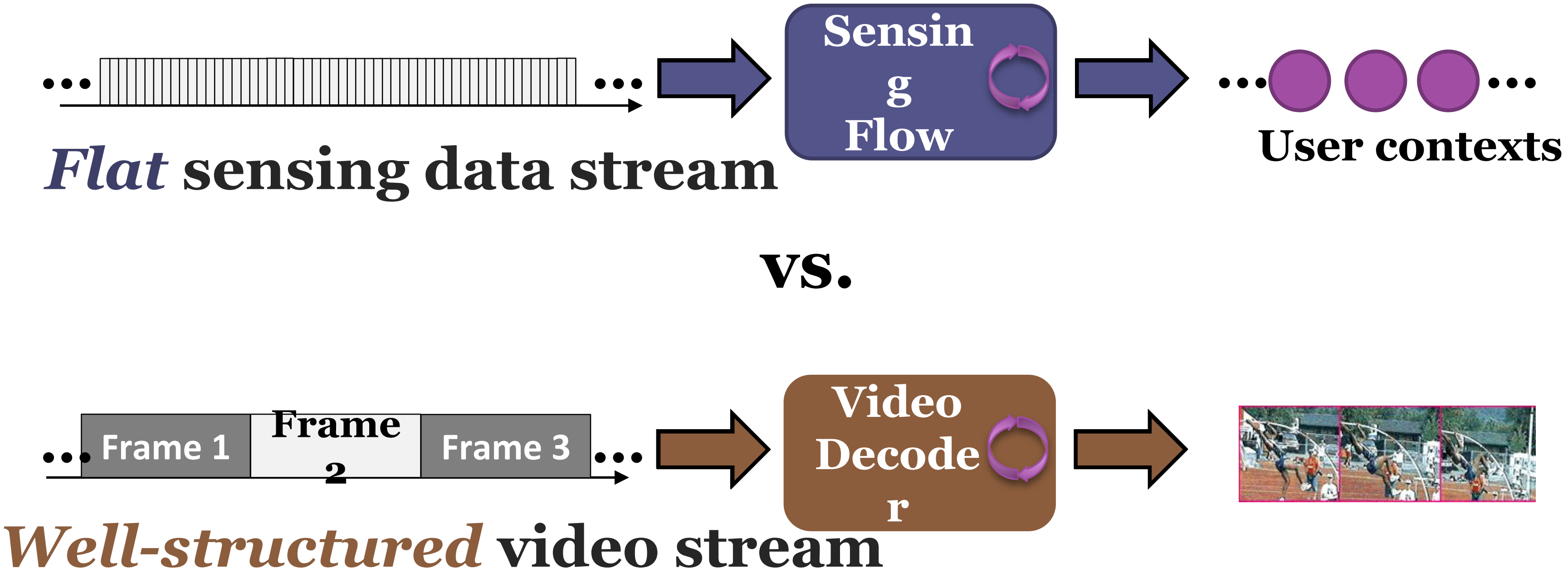
- Leverage a series of sensing and processing operations to produce a context result as a basic unit of resource allocation and scheduling
  - Inspect dataflow graphs, and identify the unit of continuous sensing data to produce a context result for each graph: *c-frame*

- Best utilize system resources under contentious situations without unnecessary waste
  - Ensure the data integrity inside assigned c-frames and the CPU time to process the frames
    - → Do not compromise the accuracy of processing results and prevent prolonged delay
    - c.f., in-frame data drop in duty cycling or downsampling
  - Eliminate unnecessary sensing and processing outside the c-frames.

- Facilitate the system to satisfy applications' service quality in the coordination process
  - Direct mapping between and resource allocation service provision

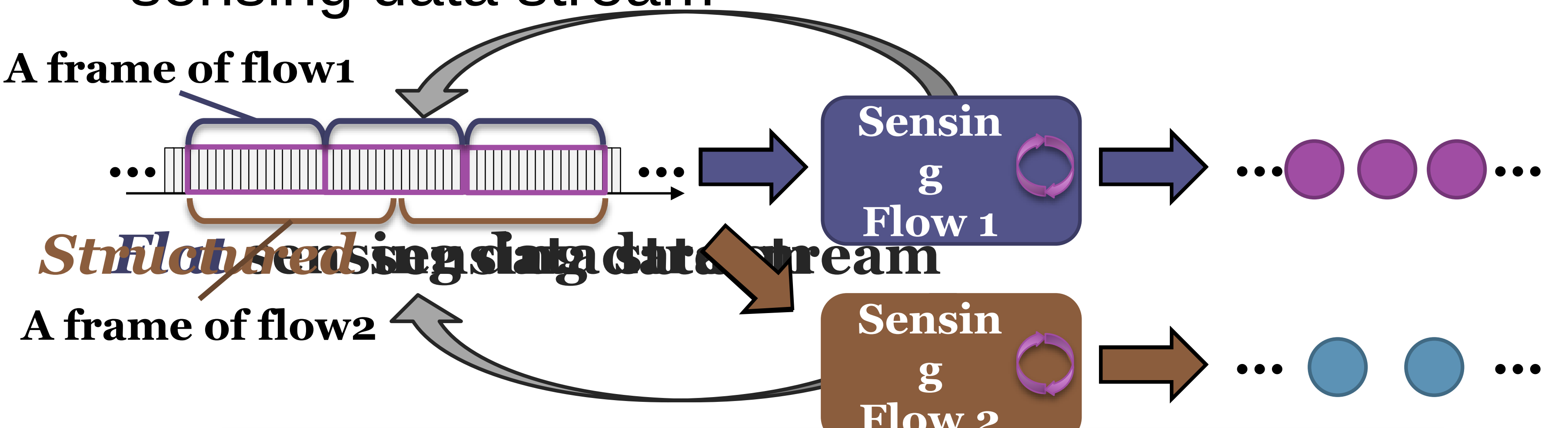
Observation

- Look into regular processing structure of sensing applications



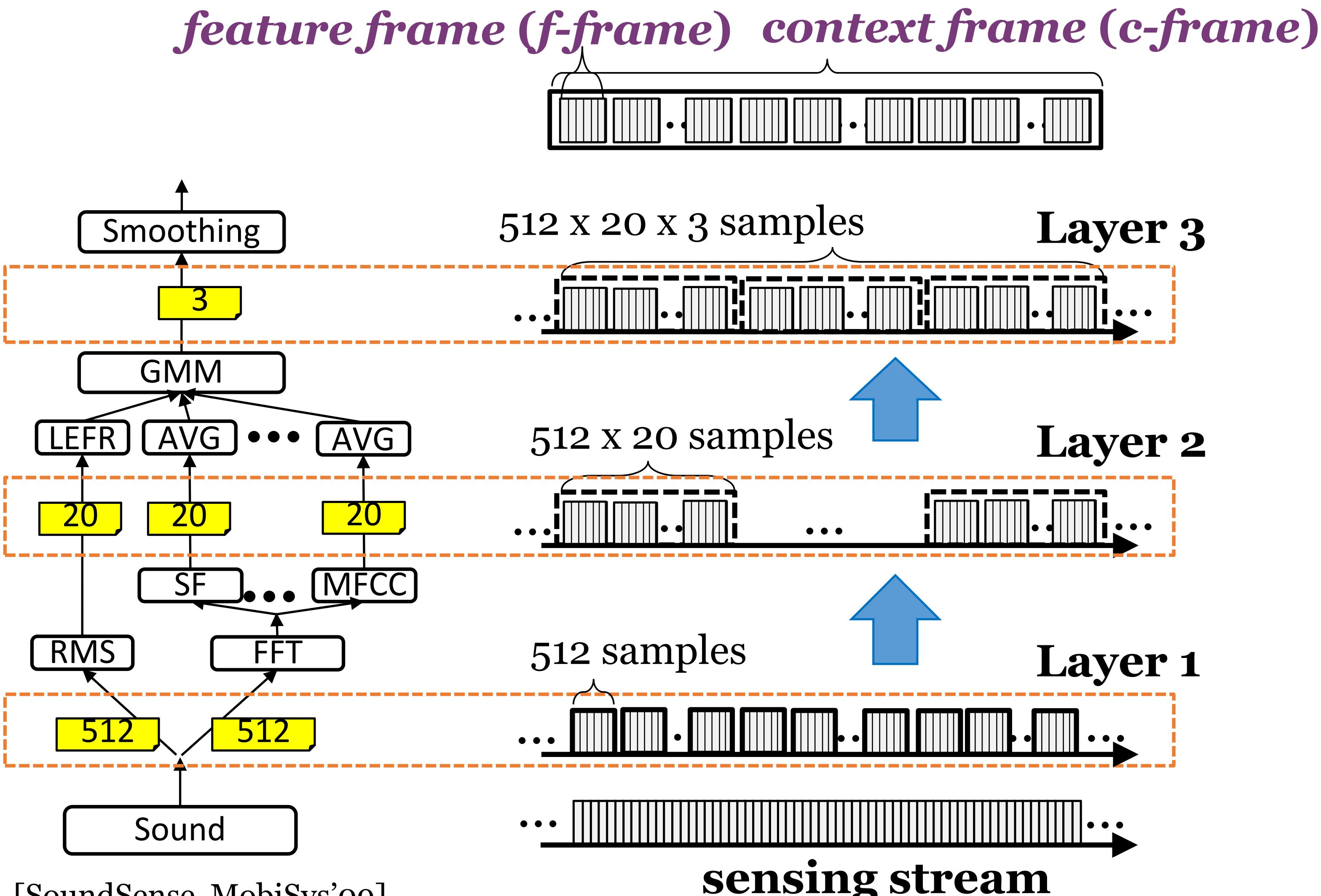
Frame Externalization

- Externalize semantic structures embedded in sensing data stream



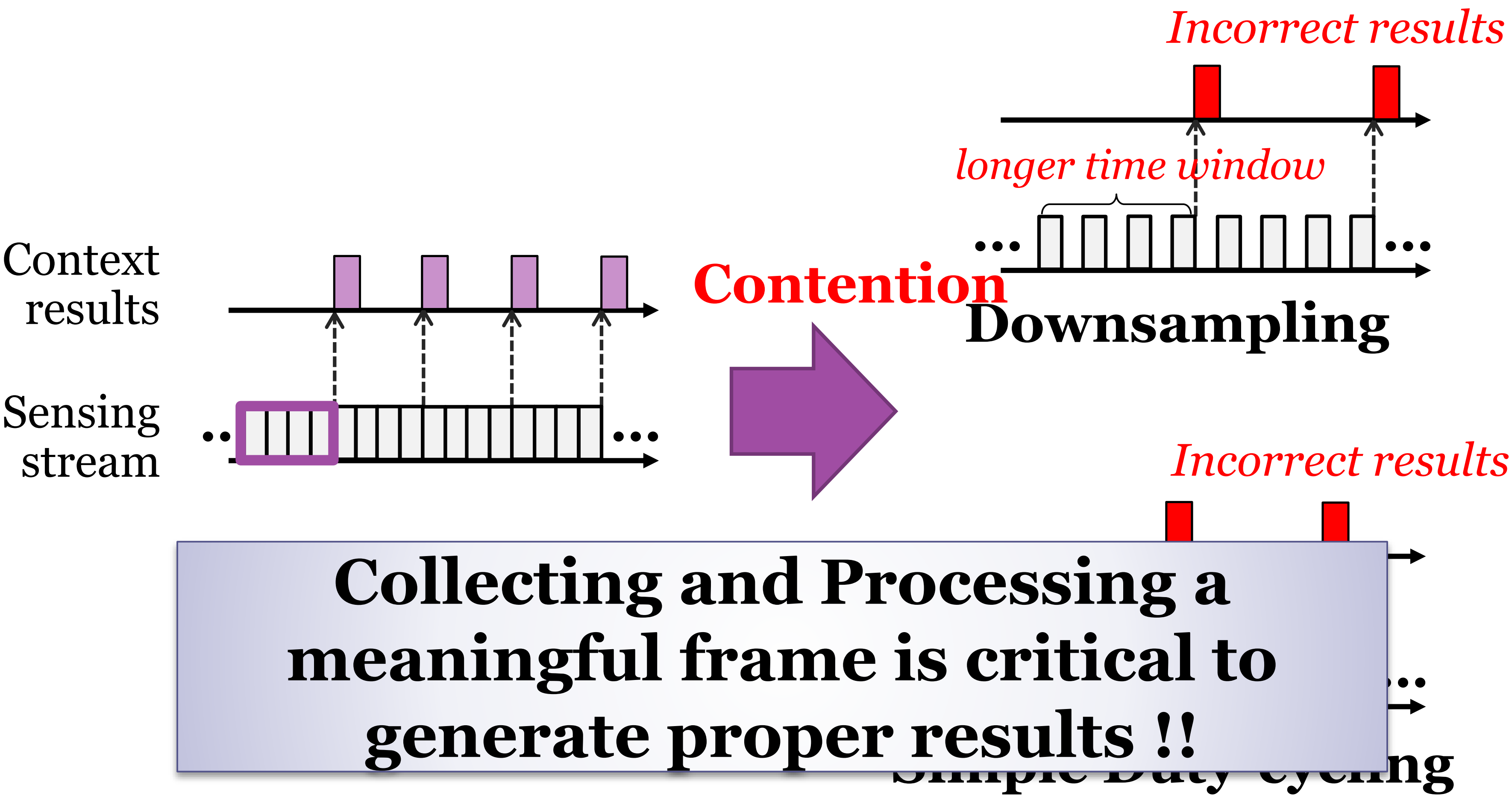
Frame externalization provides useful hints for sensing flow coordination

Multi-layered Frame Structure



[SoundSense, MobiSys'09]

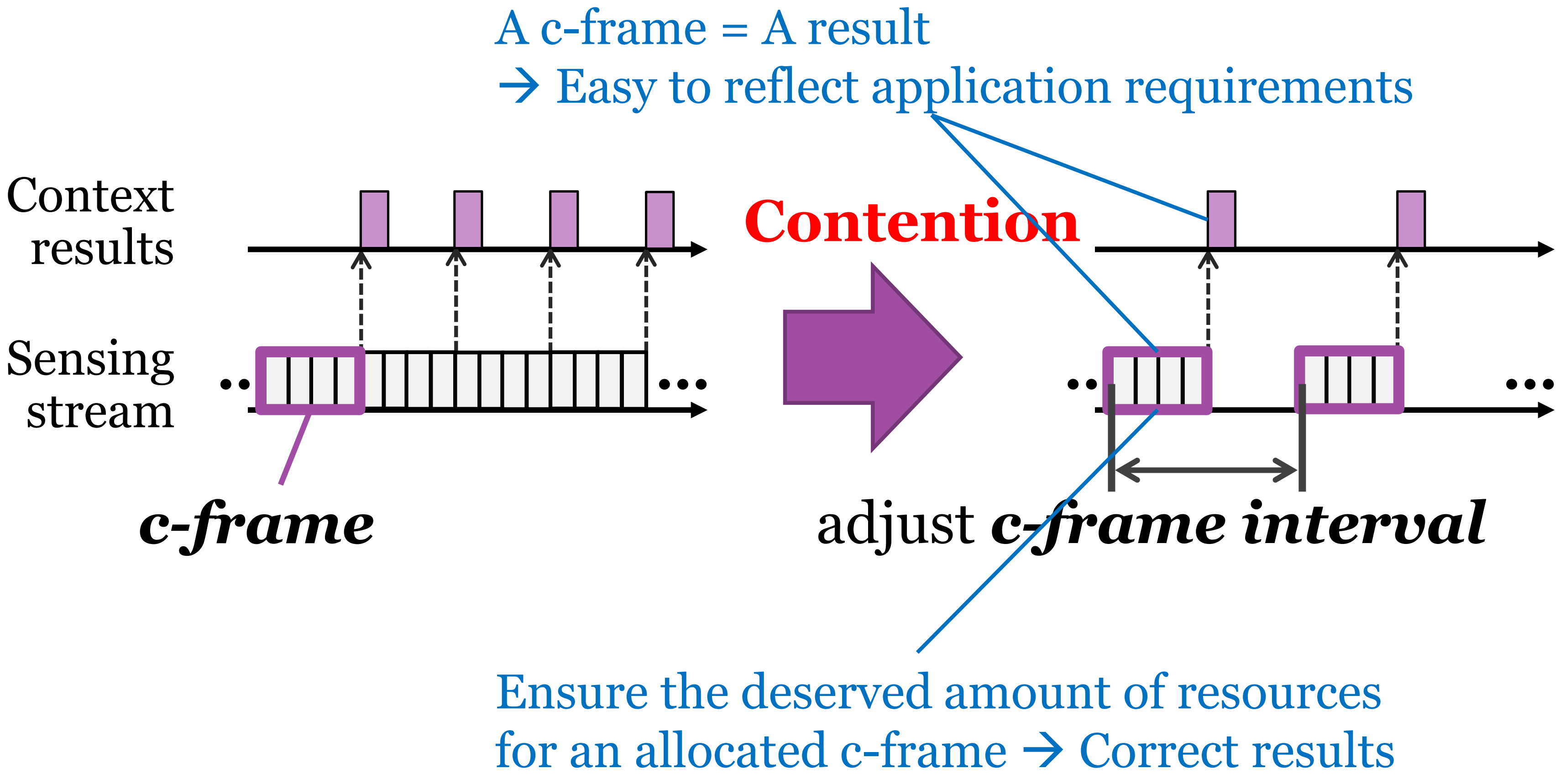
How to Adjust Sensing Apps' Resource Use?



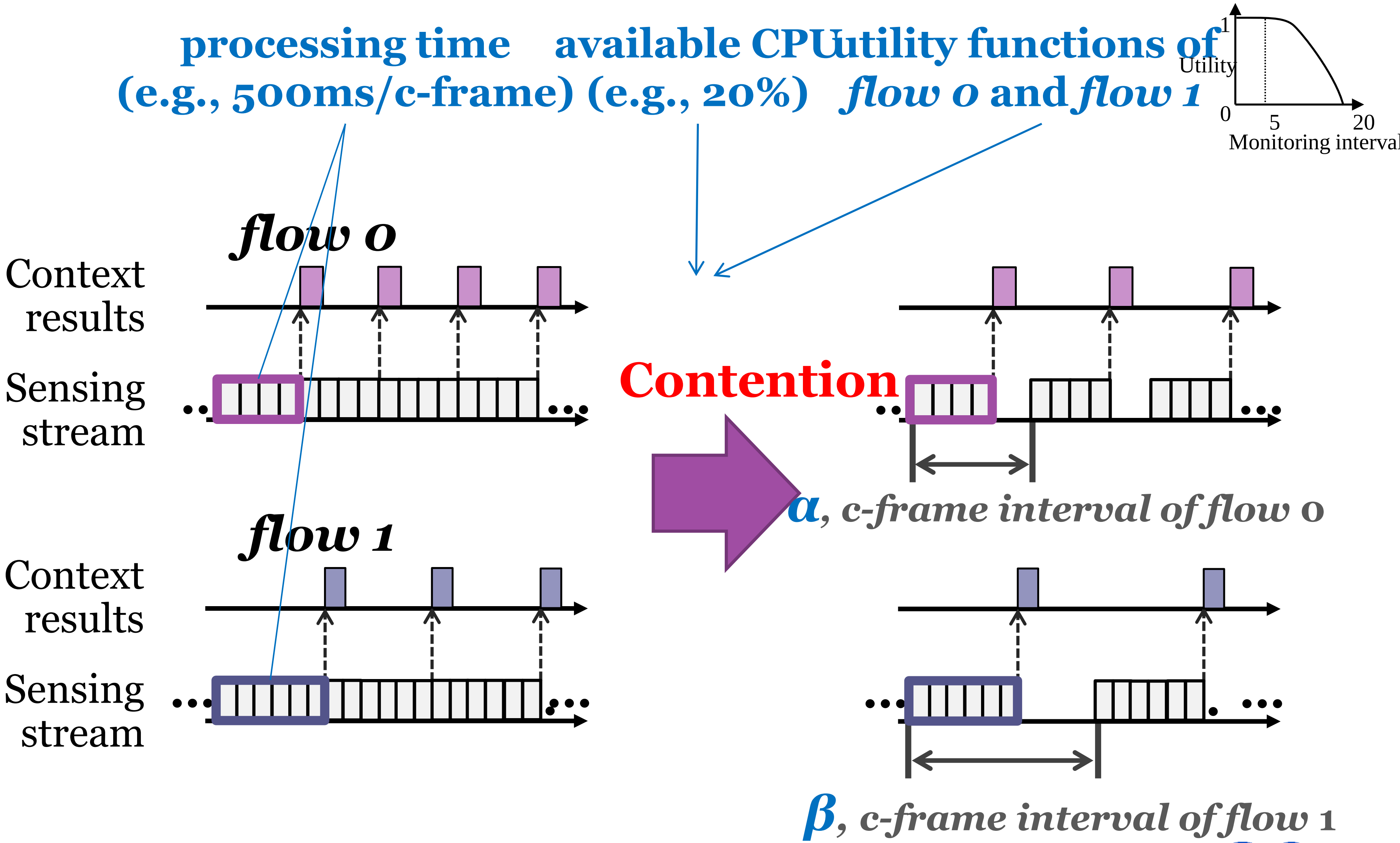
Collecting and Processing a meaningful frame is critical to generate proper results !!

c-frame-based Flow Coordination

- *c-frame* as the basic unit of resource allocation



c-frame-based Flow Coordination: Multiple Flows



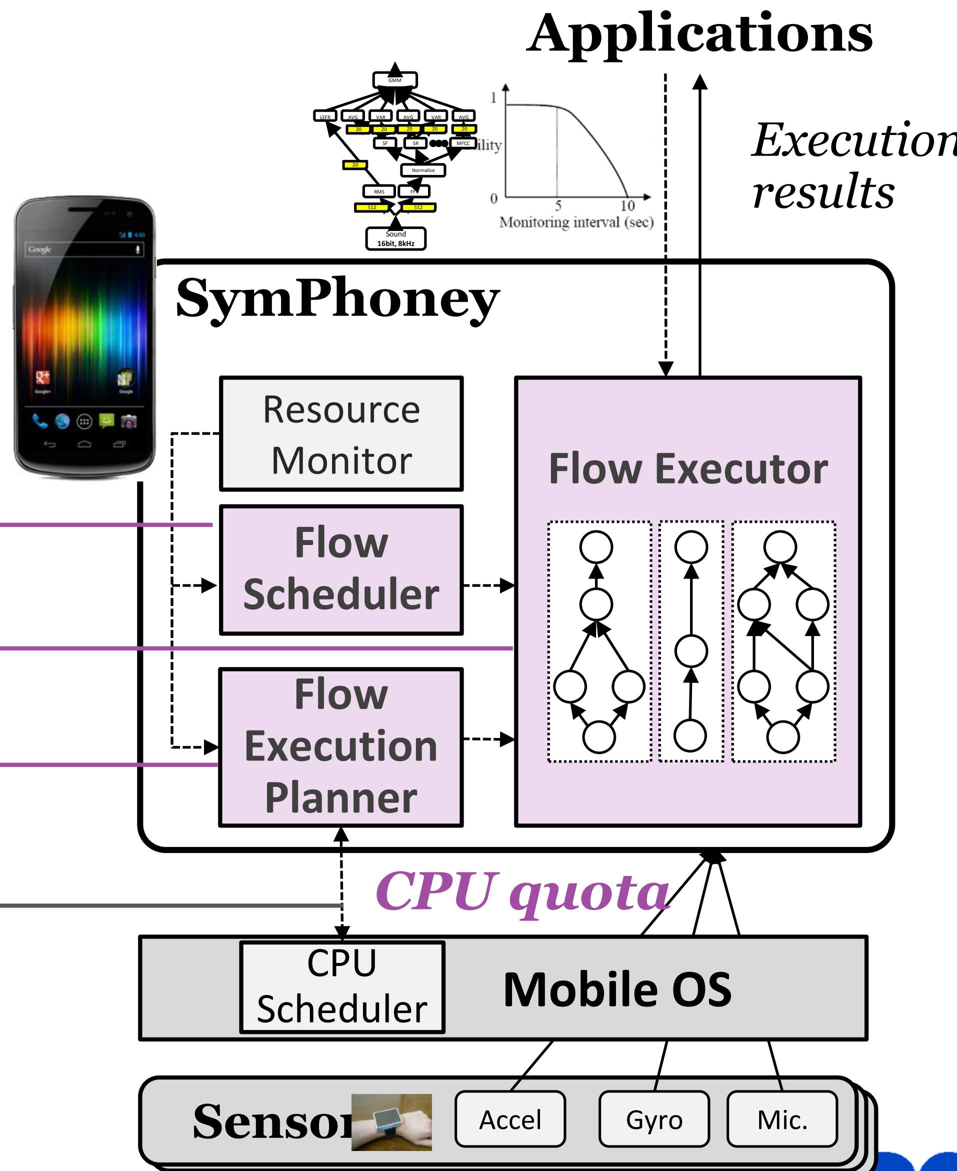
Architecture

urgency-based flow scheduling

Efficient flow execution: execution container

c-frame-based flow coordination

Resource reservation for BG sensing apps



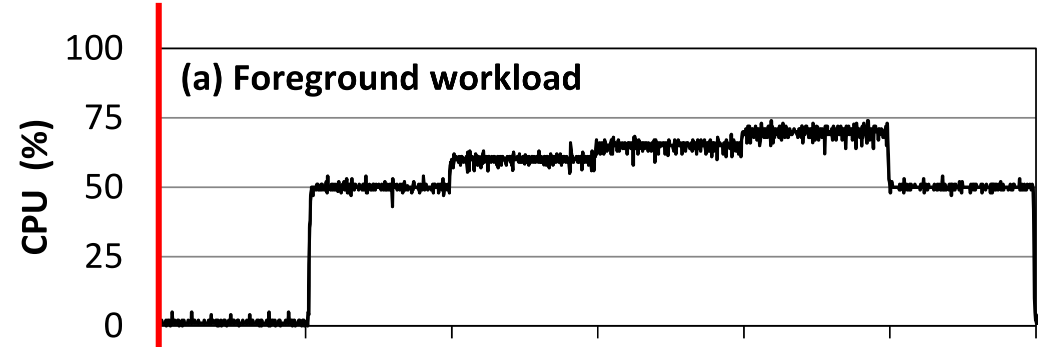
Implementation

- Implemented on Android in Java
- Run as a Android service and provide XML-based service interface
- Provide 50+ types of built-in operators commonly used in mobile sensing apps

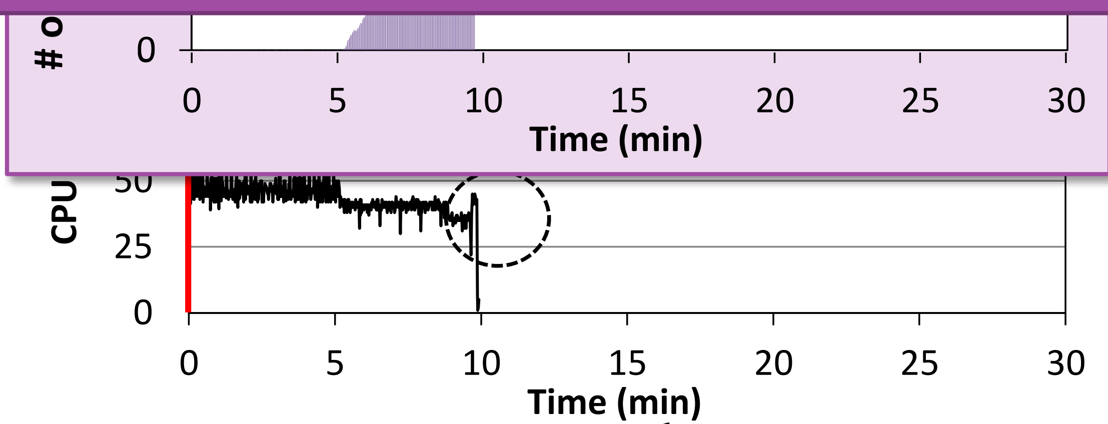
Operator types	SymPhoney built-in operators
Sensing operators	Sound, Accel., Gyro., GPS, …
Feature extractors	FFT, MFCC, RMS, Correlation, Energy, Average, Entropy, …
Classifiers	GMM, HMM, Decision tree, …

- CPU quota allocation is implemented on Android CFS scheduler
  - No kernel modification

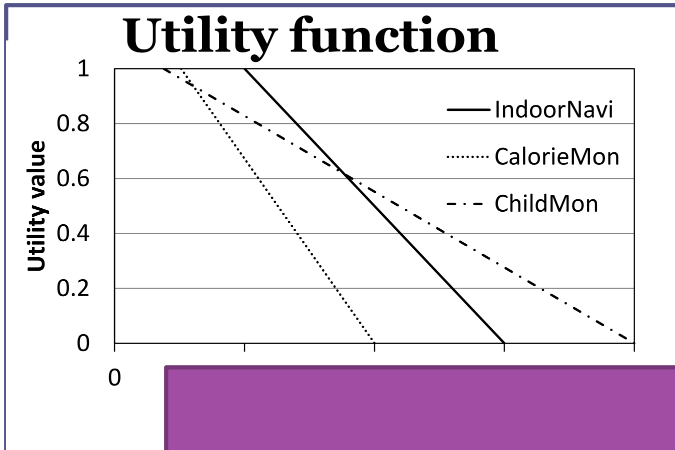
Effect of Flow Coordination (1/2)

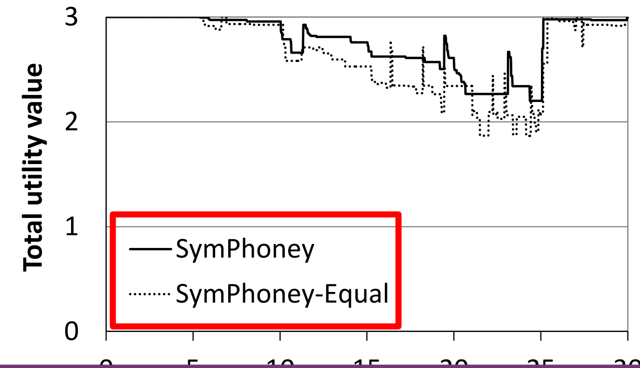


SymPhoney effectively coordinate sensing apps' resource use in changing resource situations

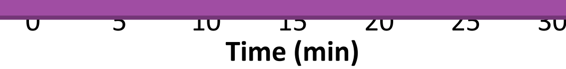


Effect of Flow Coordination (2/2)

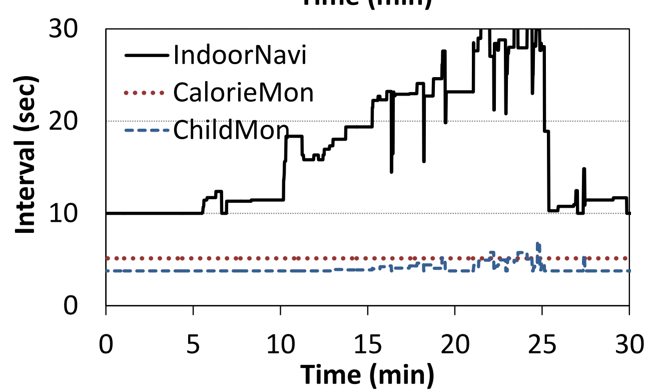




SymPhoney provides applications
with high utilities considering
their resource demand and requirements



SymPhoney-Equal
(Same policy w/ OS-Scheduler)



KAIST

NC Lab Network Computing

Conclusion (SymPhoney)

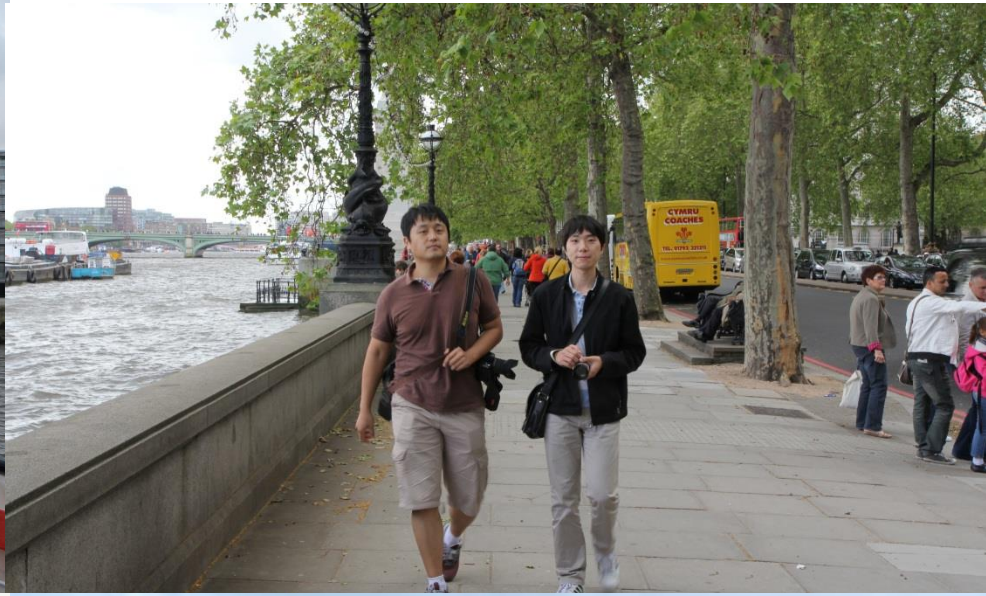
- Newly address the CPU contention problem between continuous sensing apps and with other mobile apps

- SymPhoney – a mobile sensing flow execution engine for concurrent sensing applications
  - Propose a flow-aware coordination approach
    - Develop frame-based coordination and execution methods
  - Maintain sensing apps' utilities at a reasonable level even under high contention

- Frame externalization can provide valuable hints to address various system challenges for mobile sensing applications

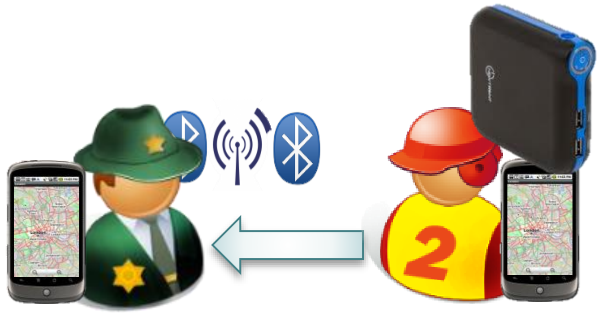
CoMon

We Travel Together. Why Everyone Sense?



Expected Power Savings

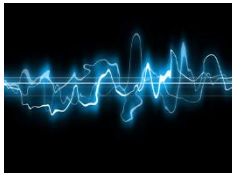
I Travel Alone



I Meet Young

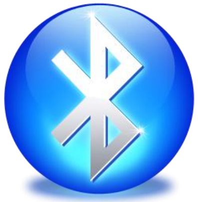
I Meet Brian

≈ 440 mW

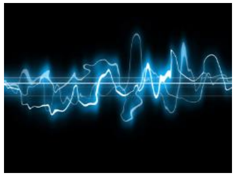


≈ 315 mW

≈ 80 mW



≈ 110 mW

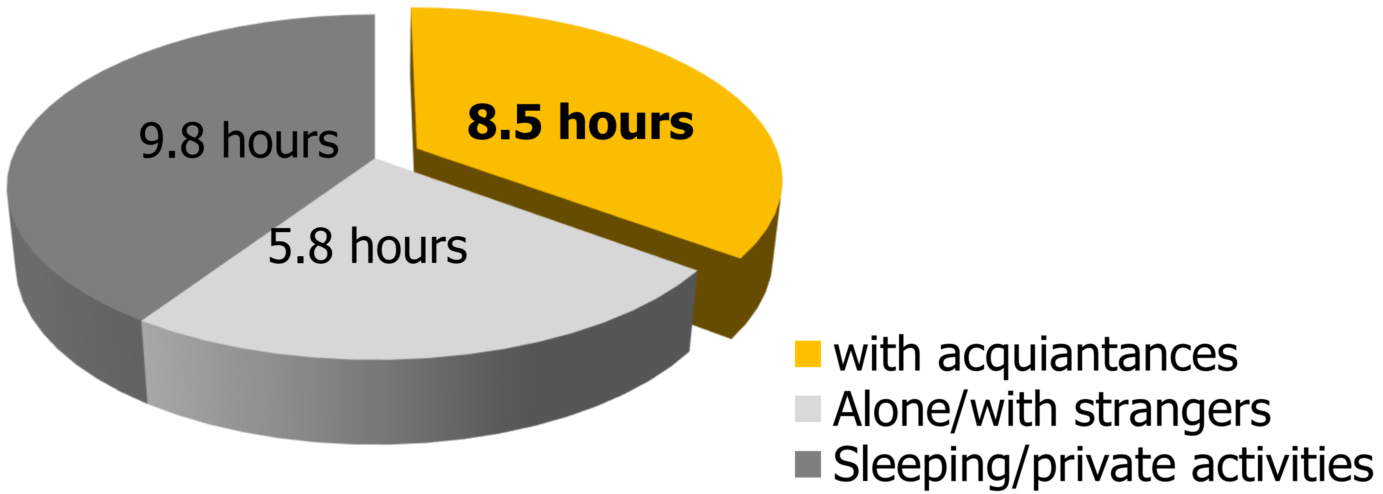


≈ 315 mW

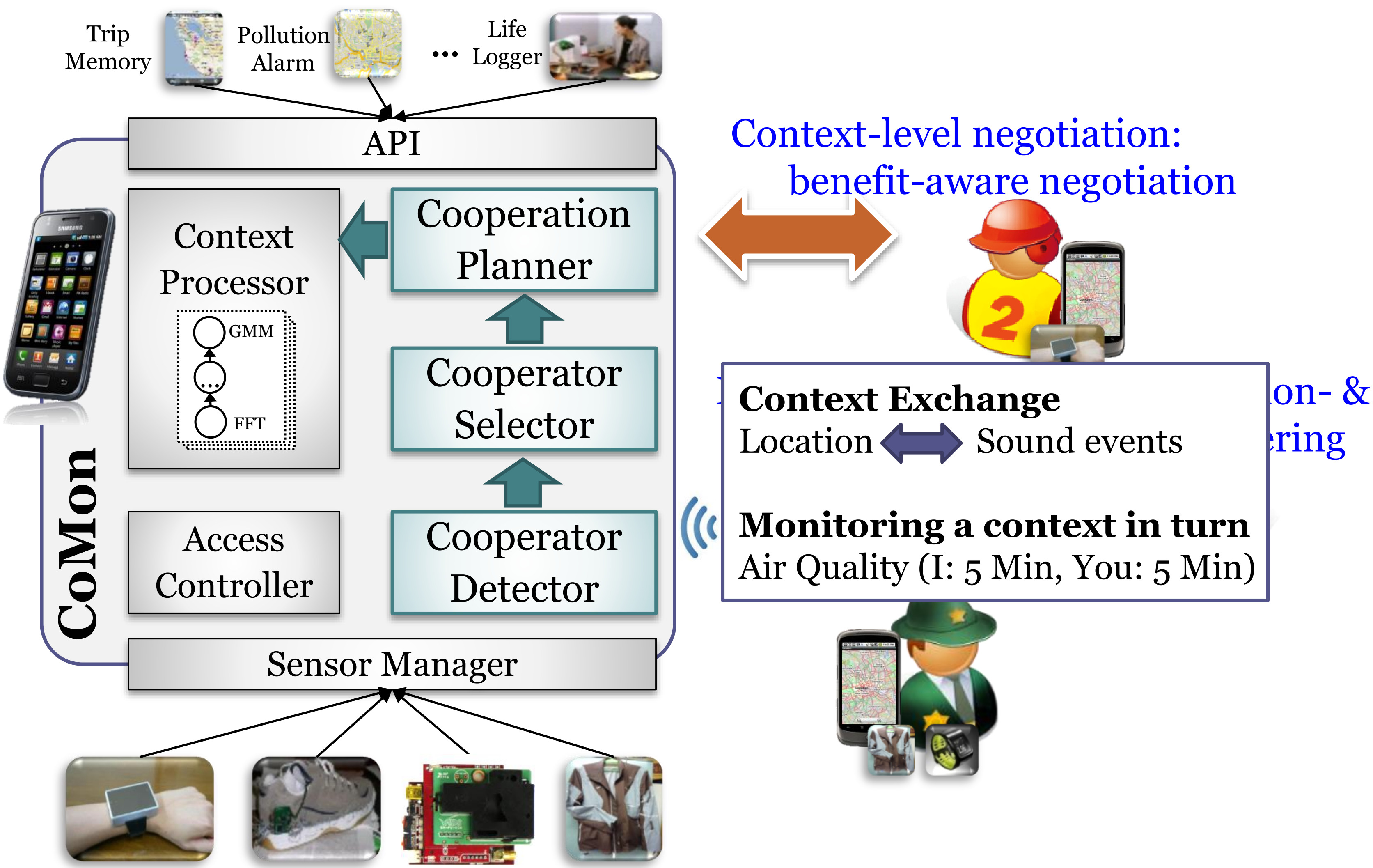
Enough Opportunities in Daily Lives !

- 8.5 hours of collocation with acquaintances (ATUS).
  - American Time Use Survey (http://www.bls.gov/tus/) :
    - Survey over 10,000 participants about what they do, for how long, with whom per day
- 47% meetings continues >1 hour. 65% do >30 Min

Time Use Per Day



CoMon Overview



Conclusion

- PAN-scale dynamic mobile computing environments
  - new personal computing environments enabling
    - continuous context monitoring and
    - personal context-aware applications
- Platform support for proactive mobile applications
  - First step toward UX-oriented mobile apps
  - A new mobile platform for the emerging sensor-rich mobile computing environments
    - Energy efficiency
    - Scalability
    - Active resource use orchestration
  - support a number of concurrent applications with highly scarce and dynamic resources
  - active coordination between multiple applications and multiple underlying resources

Thanks a lot! Questions

Juehwa Song

Dept. of CS
KAIST

junesong@cs.kaist.ac.kr
http://nclab.kaist.ac.kr