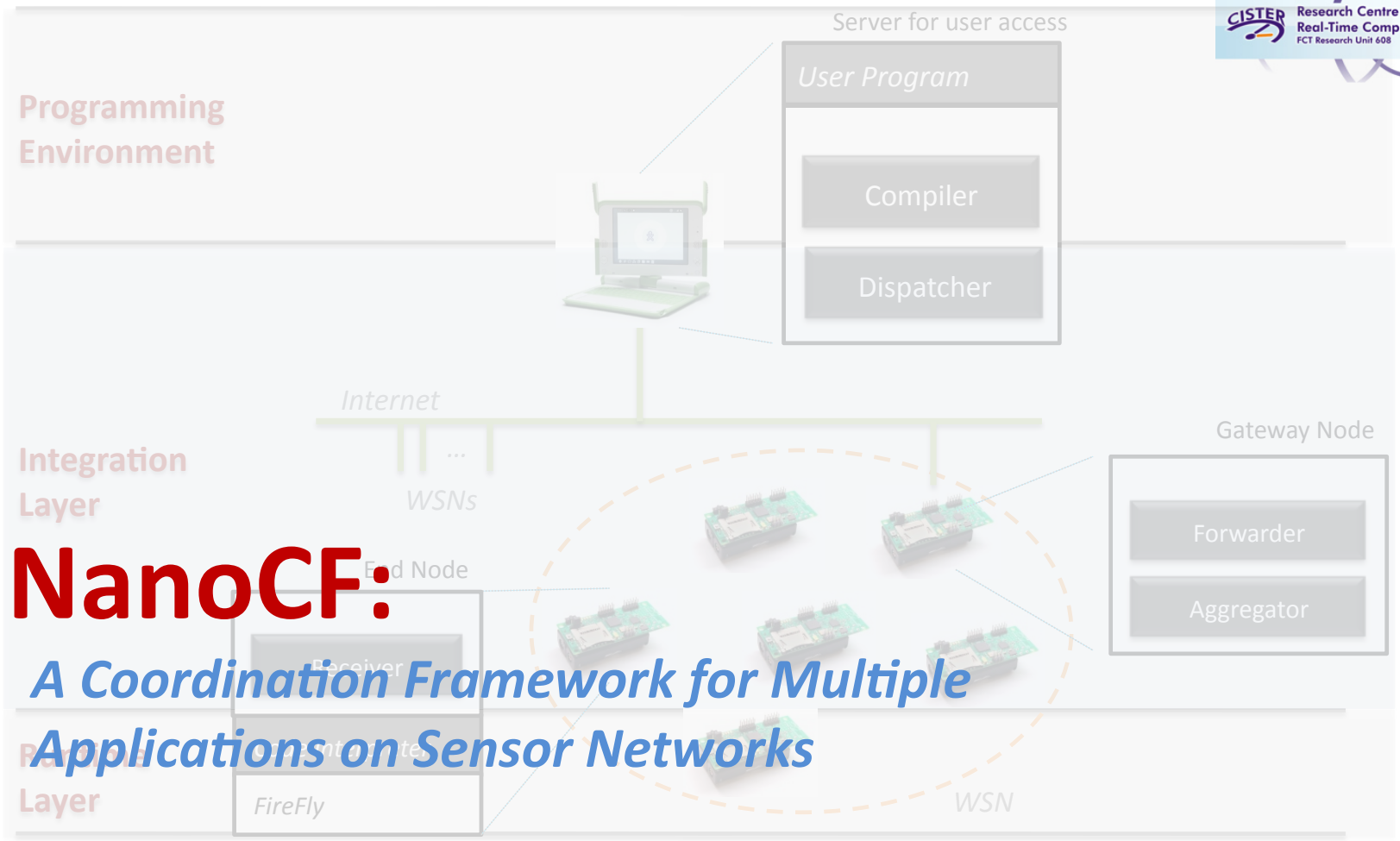


NanoCF:

A Coordination Framework for Multiple Applications on Sensor Networks

Vikram Gupta, Eduardo Tovar, Luis Miguel Pinho
Junsung Kim, Karthik Lakshmanan, Raj Rajkumar

Macro-programming support

- **In-network programming**
- **Why?**
  - Usability
  - Lack of Technical Expertise with non CS people
  - Cost of re-program-ability
  - Faster deployment
  - Heterogeneous Hardware/Software
- **Consider a network with hundreds of nodes**
  - Few minutes per node can mean hours
- **More "Qualitative" advantages**

Sensor Network as Infrastructure

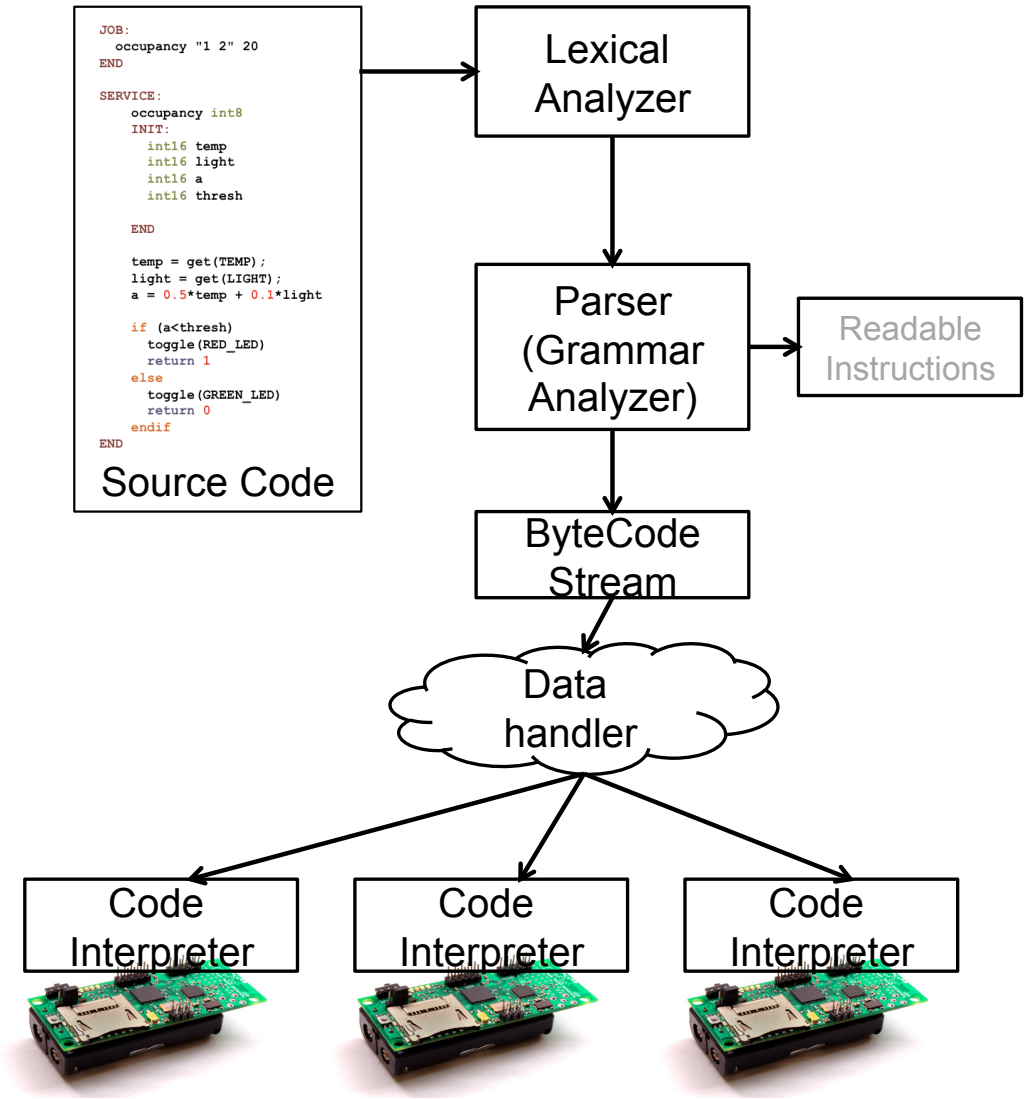
Occupancy Checking and Room Climate Control

- Several applications on same sensor network
- Geographically distributed sensor network
- Limited flexibility and usability

Challenges for Concurrent applications

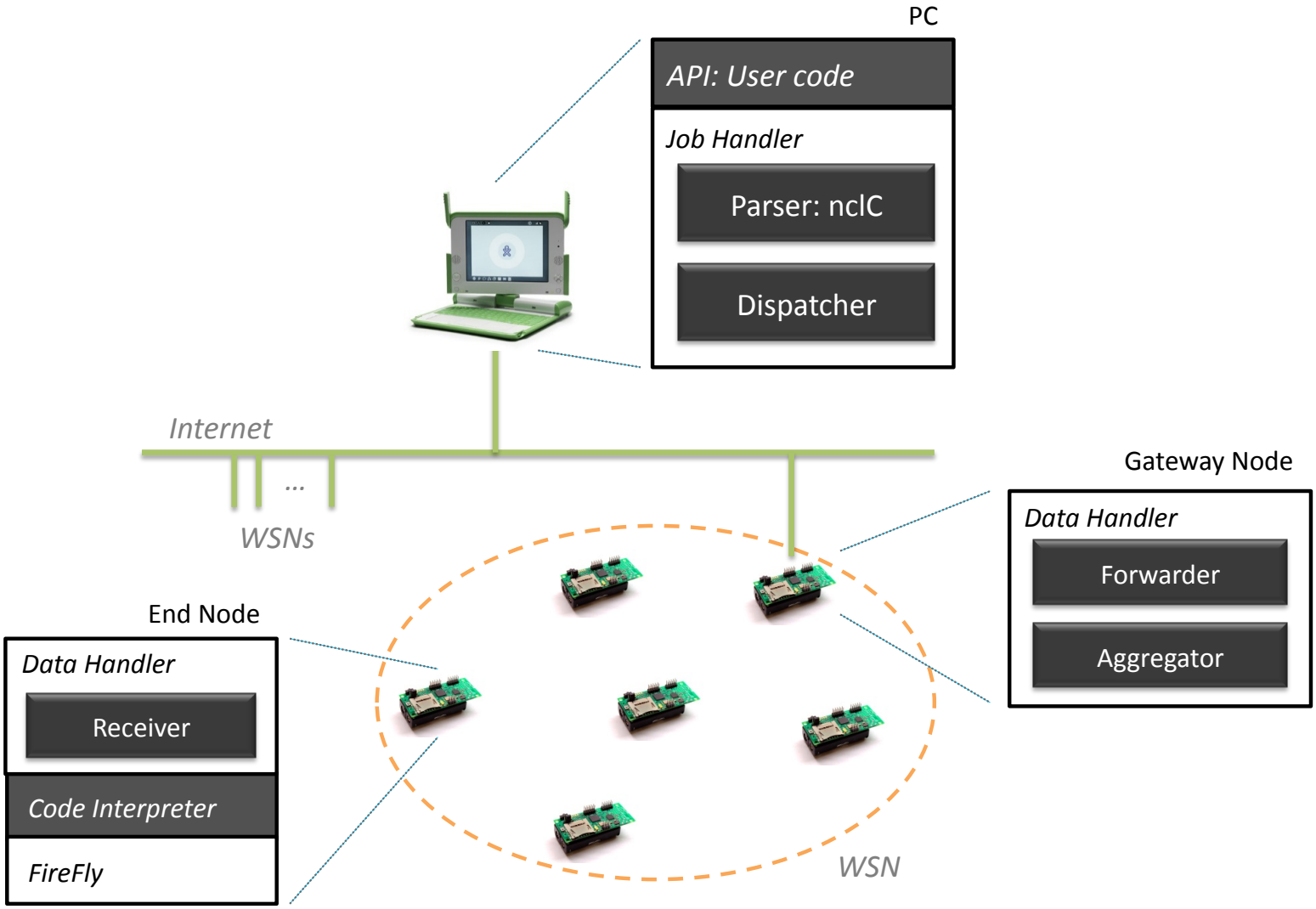
- **User Interface**
  - Database queries, virtual machine etc..
- **Operating System Support**
- **Packets through multiple applications over multi-hop network**
- **Data Aggregation**
- **Minimizing the overhead**
  - Frequency of Processor and Radio On/Off
  - Network flooding
  - Seamless backend handling
- **Tradeoff between Control and Abstraction**

Macro Programming System

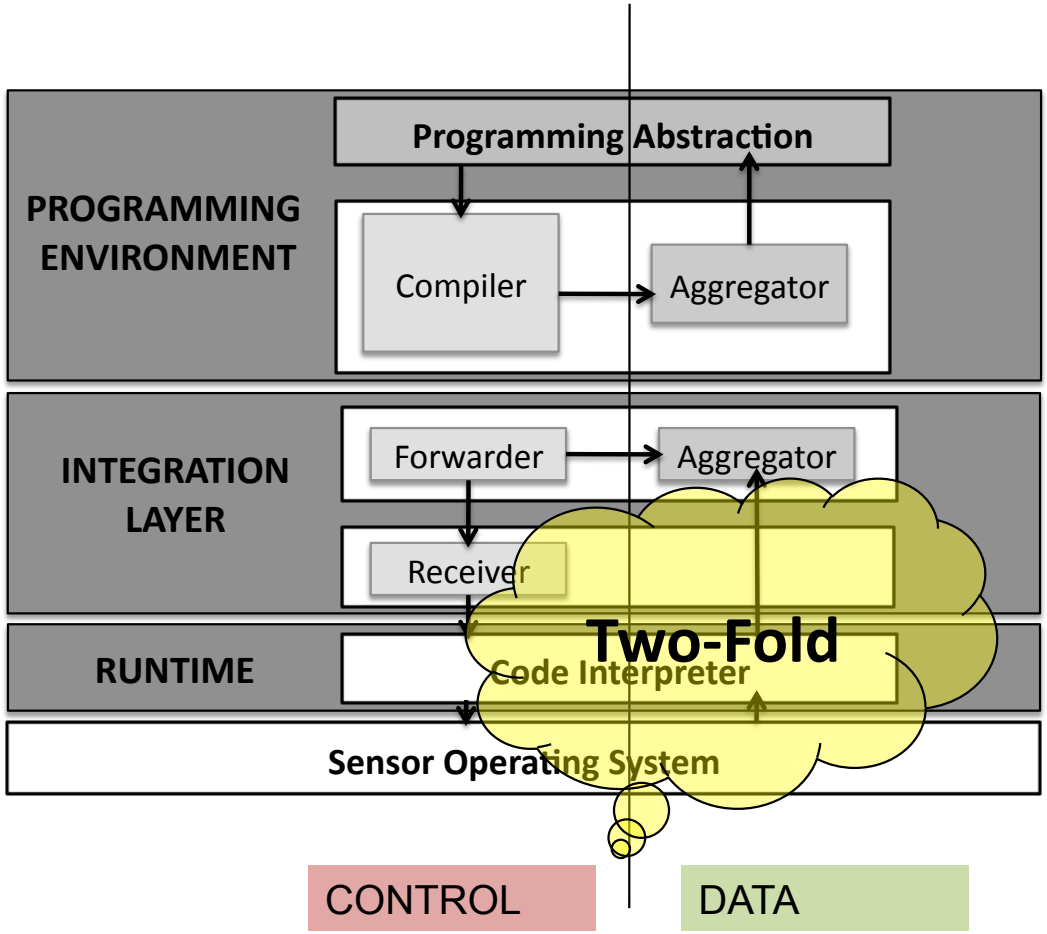


- nanoCL: Small **C**omposible **L**anguage for Sensor Networks
- Abstracts away from lower-level details
- Supports various data-types and library functions
- Independent of the type of sensor nodes

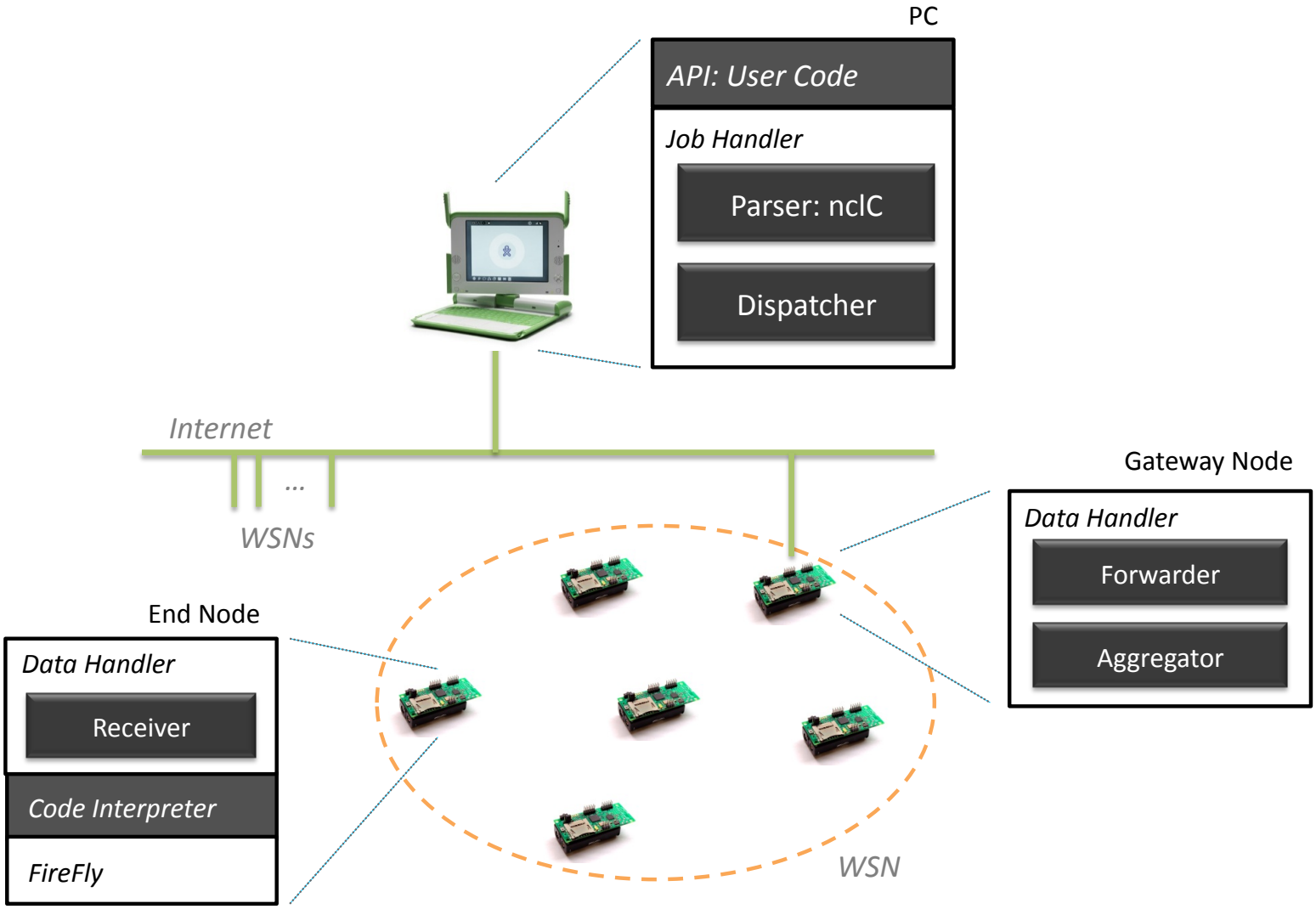
System Architecture



Control and Data Flow

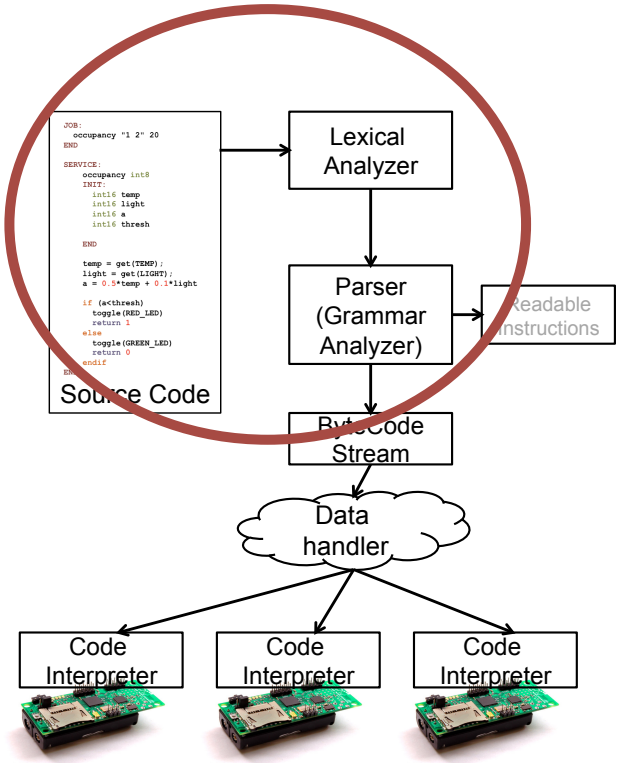


System Architecture Outline



sMapReduce Programming Abstraction

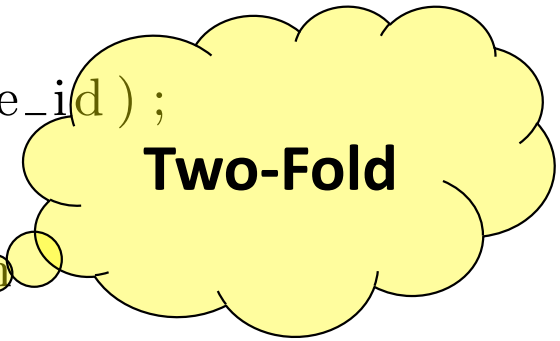
- **Natural-fit to sensor network operation**
  - Map the "functionality" to sensor node
  - Gather the data through the network tree (Reduce)
- **Inspired from Google's MapReduce**
- **Key Features**
  - Balanced abstraction and control
  - Easy debugging
  - Two-fold operation



Simple Temperature Collection Example

```
1 smap(service_name, list_of_nodes, period){
2   for each node in list_of_nodes
3     temp_value = gets(TEMP);
4     smap_emit(temp_value, node_id);
5   end
```

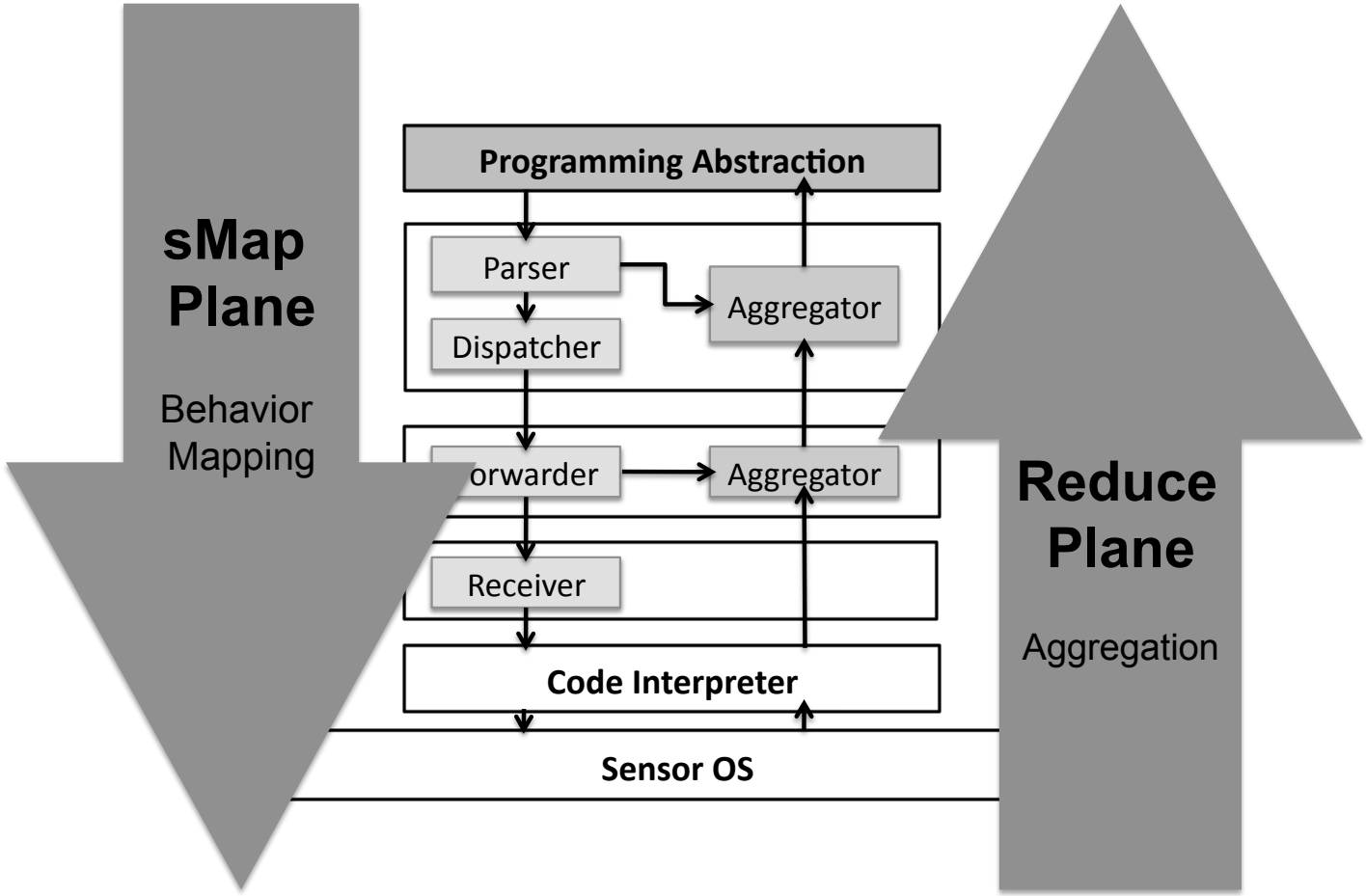
(a) sMap Function



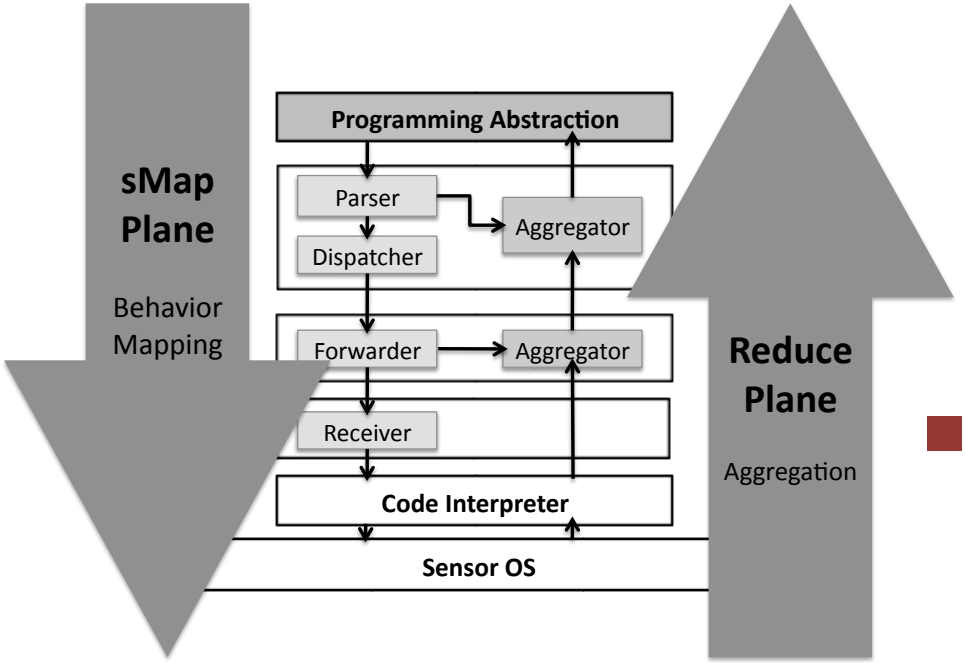

```
1 reduce(data, list_of_nodes){
2   for each node in INNER.list_of_nodes
3     sum += data.temp_value; //AGGREGATION
4   end
5   return sum;
6 }
```

(b) Reduce Function

Operation to UI Correlation



sMap and Reduce planes

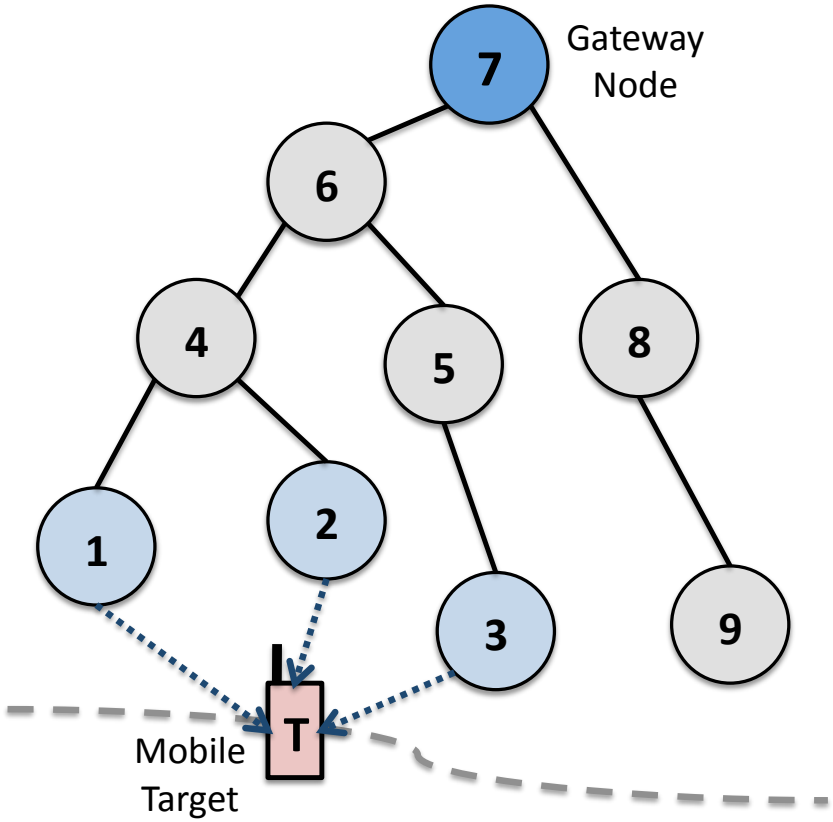


- Left plane handles mapping the behavior
- Right plane handles the aggregation of data through the network

Target tracking application

```
1 smap(target_track, list_of_nodes, period){
2   for each node in list_of_nodes
3     rssi_v = get(RSSI);
4     ts = get(time);
5     smap_emit(rssi_v, ts, node_id, loc);
6   end
```

```
1 reduce(data, list_of_nodes){
2   for each node in INNER.list_of_nodes
3     if(data.loc != NULL)
4       return data.loc;
5     else
6       if(max(ts)-min(ts)<=win
7           && size(data.rssi_v) >= 3)
8         triangulate(rssi_v, loc);
9       else
10        return data;
11      end
12    end
13  end
14 }
```



Write-ability

- **Simple C like syntax**
- **Library functions**
  - `gets()` for accessing sensor data
  - Arithmetic operations
  - int/uint data types
  - `set(), get(), toggle()` for GPIO pins
  - for and while loops
  - `if/else` constructs
  - `return` values to collect data

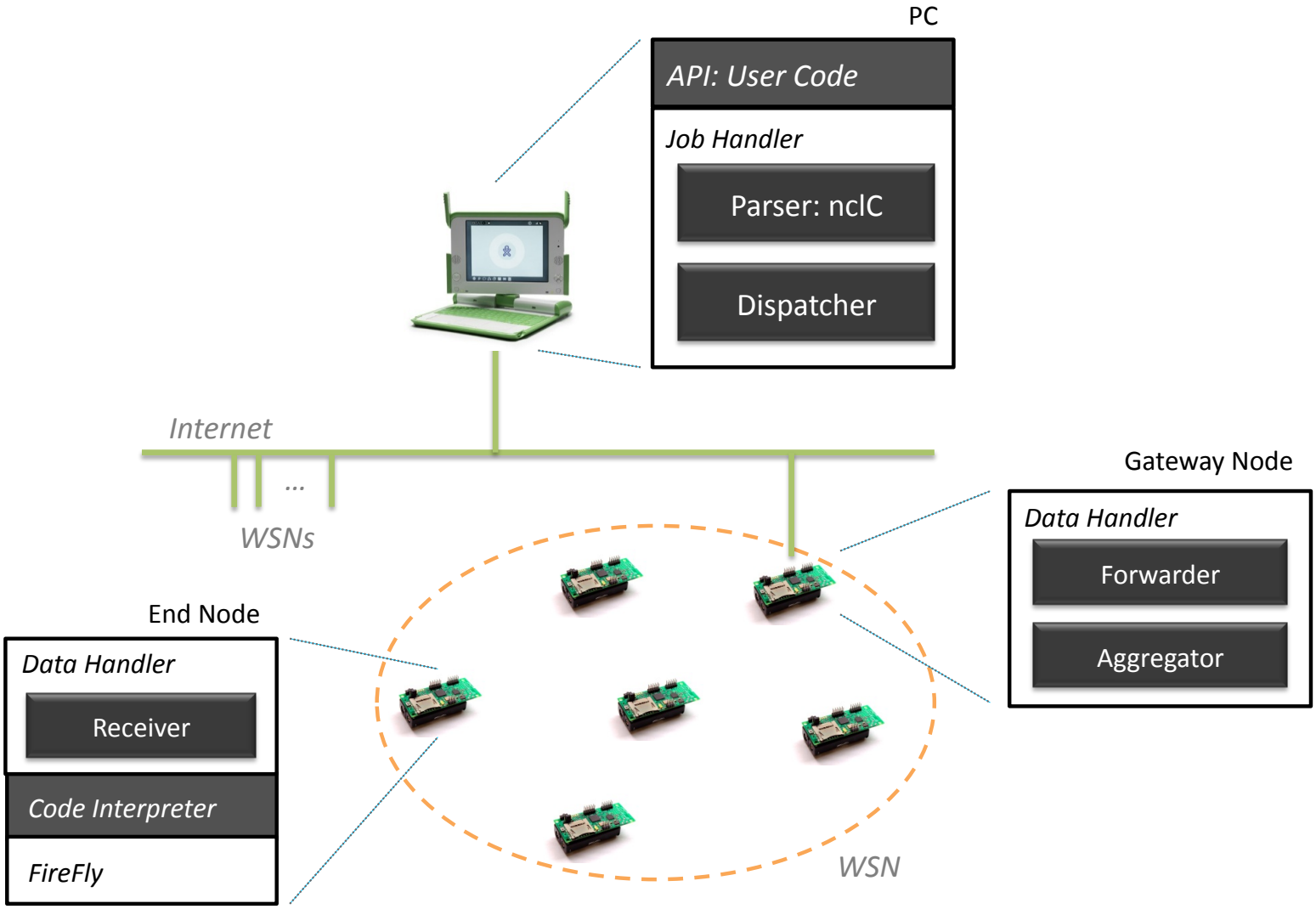
Source Lines of Code Comparison

Application	NanoCF	Operating System
Temperature Collection	5	80
Occupancy Monitoring	20	205
Target Tracking	20	~ 300 - 400

nanoCL Code	Corresponding Assembly	ByteCodes Generated by nanoCL Compiler

JOB:
  dummyservice "1 2 3 4 5" 100 MIN
ENDJOB
 	
SERVICE:
  dummyservice int8 int8
  INIT:
  int8 aa
  int8 bb
  int8 cc
  ENDINIT
  aa = gets(TEMP)
  bb = gets(LIGHT)
  clt (LED RED)
  cc = (bb/100) + (aa/100)
  if(cc > 15)
  set(LED RED)
  print(cc)
  endif
  wait(100)
ENDSERVICE
	No of Instructions:
	35
	  SECTION INIT
	int8 a (aa)
	int8 b (bb)
	int8 c (cc)
	int16 d
	int16 e
	int16 f
	int16 g
	int16 h
	int16 i
	int16 j
	  ENDINIT
	  SECTION SERVICE
	GETS TEMP aa
	GETS LIGHT bb
	CLR LED RED
	AEQ d 100
	DIV e b d
	AEQ f 100
	DIV g a f
	ADD h e g
	MOV c h
	AEQ i 15
	GT c i
	IF
	GOTO 11
	LABEL 12
	AEQ j 100
	WAIT j
	ENDSERVICE
	REPEAT 0x00 0x64
	LABEL 11
	SET LED RED
	PRINT c
	GOTO 12
		
		0x56, 0x58, 0xff, 0xff,
		0x5c, 0x60, 0x61, 0xff,
		0x5c, 0x60, 0x62, 0xff,
		0x5c, 0x60, 0x63, 0xff,
		0x5c, 0x61, 0x64, 0xff,
		0x5c, 0x61, 0x65, 0xff,
		0x5c, 0x61, 0x66, 0xff,
		0x5c, 0x61, 0x67, 0xff,
		0x5c, 0x61, 0x68, 0xff,
		0x5c, 0x61, 0x69, 0xff,
		0x5c, 0x61, 0x6a, 0xff,
		0x59, 0xff, 0xff, 0xff,
		0x56, 0x5a, 0xff, 0xff,
		0x30, 0x61, 0x90, 0x00,
		0x30, 0x62, 0x96, 0xff,
		0x41, 0x95, 0x03, 0xff,
		0x16, 0x64, 0x00, 0x64,
		0x1a, 0x65, 0x62, 0x64,
		0x16, 0x66, 0x00, 0x64,
		0x1a, 0x67, 0x61, 0x66,
		0x0d, 0x68, 0x65, 0x67,
		0x17, 0x63, 0x68, 0xff,
		0x16, 0x69, 0x00, 0x0f,
		0x11, 0xff, 0x63, 0x69,
		0x51, 0xff, 0xff, 0xff,
		0x53, 0x11, 0xff, 0xff,
		0x54, 0x12, 0xff, 0xff,
		0x16, 0x6a, 0x00, 0x64,
		0x44, 0x6a, 0xff, 0xff,
		0x5b, 0xff, 0xff, 0xff,
		0x45, 0xff, 0x00, 0x64,
		0x54, 0x11, 0xff, 0xff,
		0x40, 0x95, 0x03, 0xff,
		0x31, 0x63, 0xff, 0xff,
		0x53, 0x12, 0xff, 0xff,

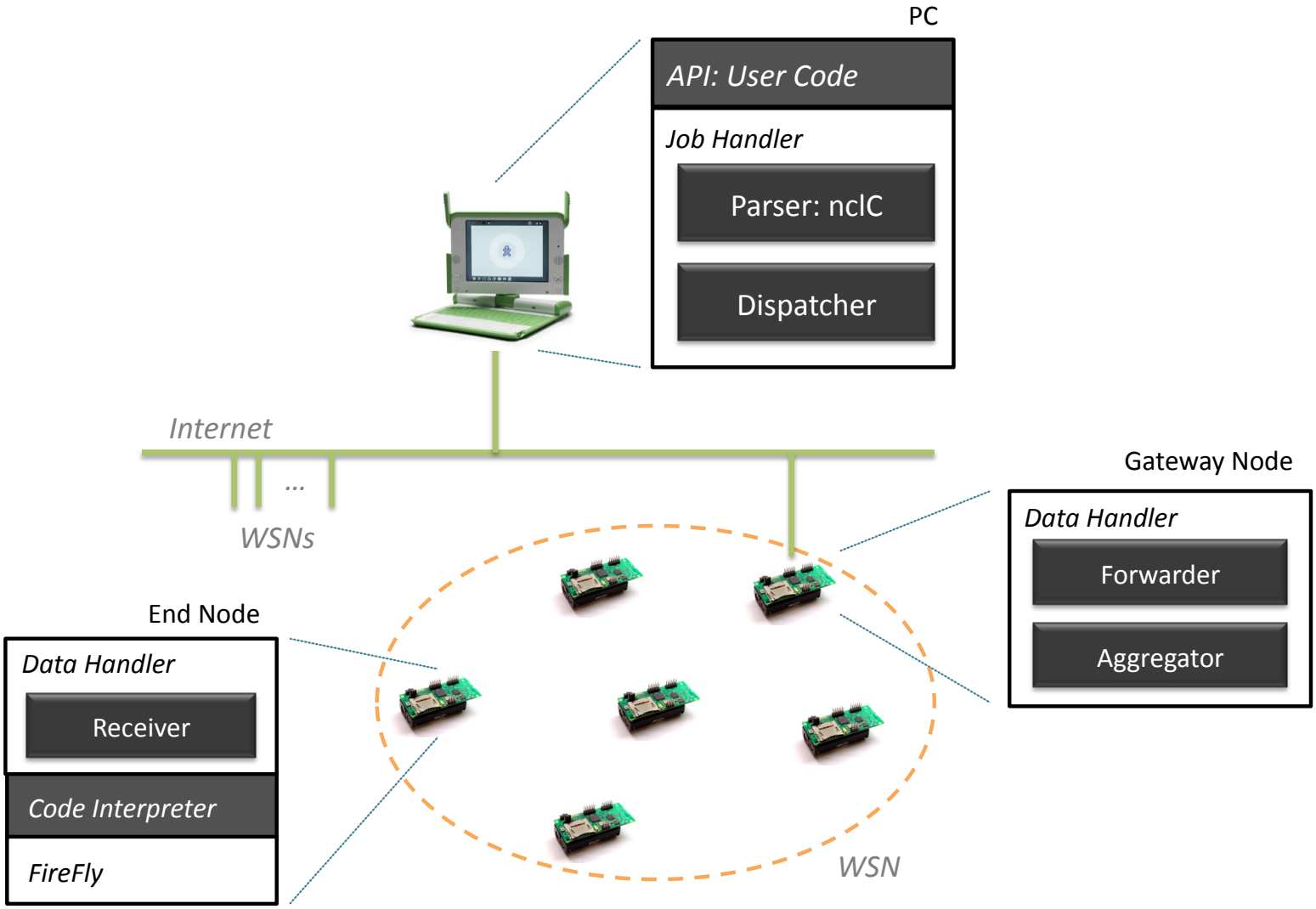
System Architecture Outline



Data Handler Functions and Features

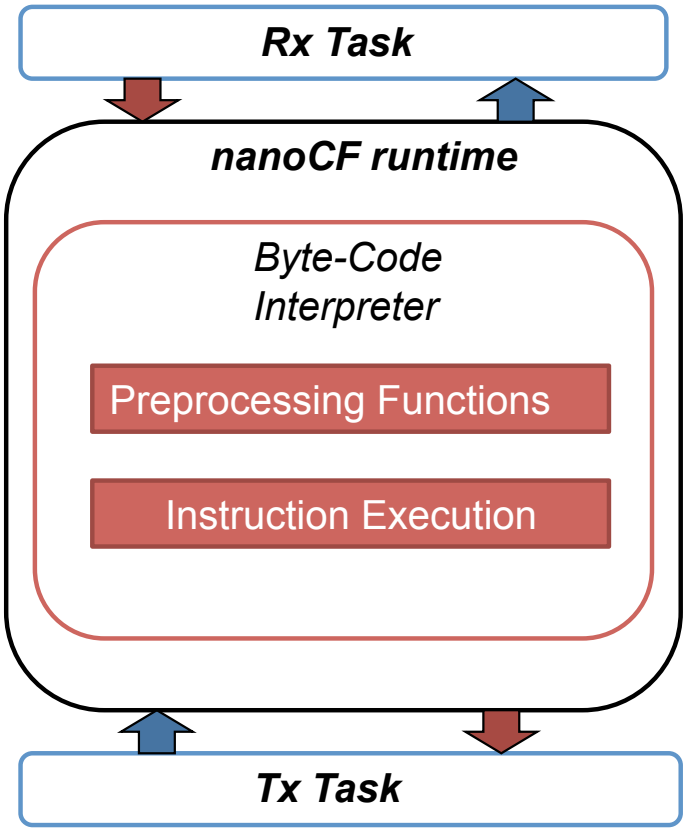
- **Functionalities**
  - Byte-code transfer
  - Data transfer and aggregation
  - Radio resource management
- **Features**
  - Routing table management
  - Fault-tolerant packet delivery
    - Retransmission
    - Random back-off delay between responses
  - Application management
    - Tracking application transaction
    - Starting and terminating applications

System Architecture Outline



Code Interpreter

- Rx Task re-arranges received packets based on sequence
- Runtime pre-processes symbols and labels in the stack
- Interprets the instructions, evaluates values
- Sends the response value back to the gateway



Challenges for Concurrent applications

- User Interface
  - Database queries, virtual machine etc..
- Operating System Support
- Packets through multiple applications over multi-hop network
- Data Aggregation

- **Minimizing the overhead**
  - Frequency of Processor and Radio On/Off

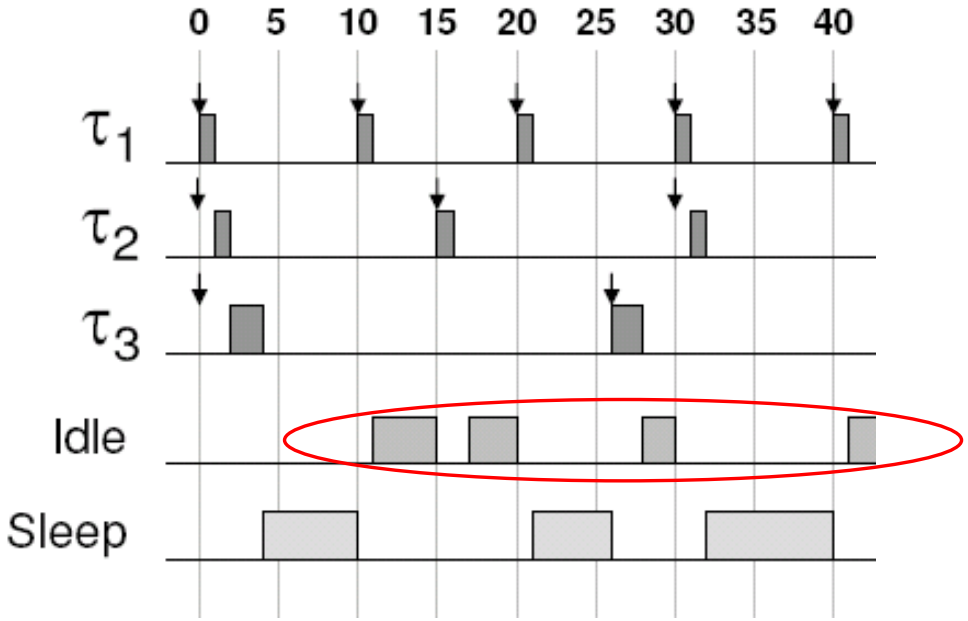
  - Network flooding
  - Seamless backend handling
- Tradeoff between Control and Abstraction

Task and Packet Scheduling

- **Typical Microprocessor operation states:**

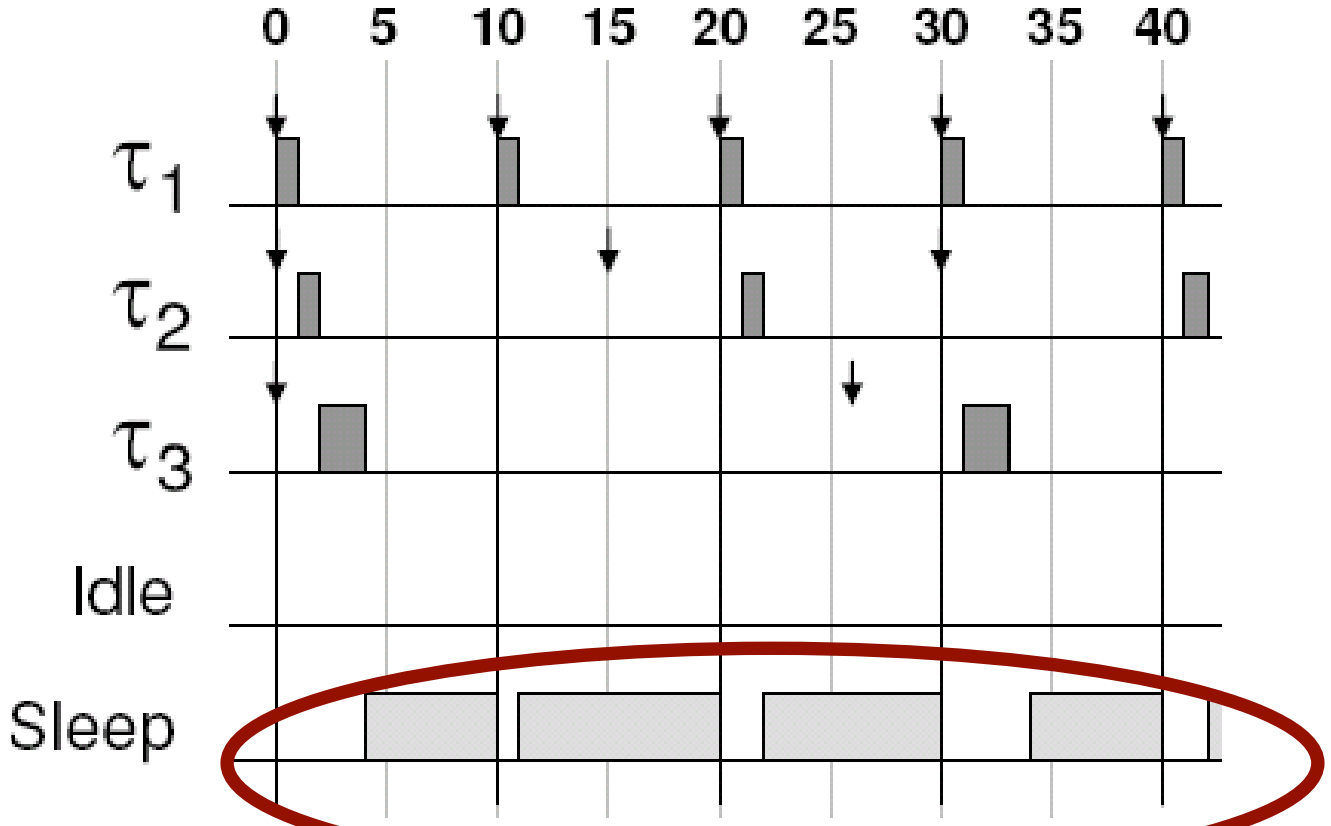
Power State	Power (mW)	Upward Transition Time
Active	30 mW	n/a
Idle	6 mW	6 μs
Sleep	5 μW	5 ms

τ_1 $(1, 10)$
τ_2 $(1, 15)$
τ_3 $(2, 25)$



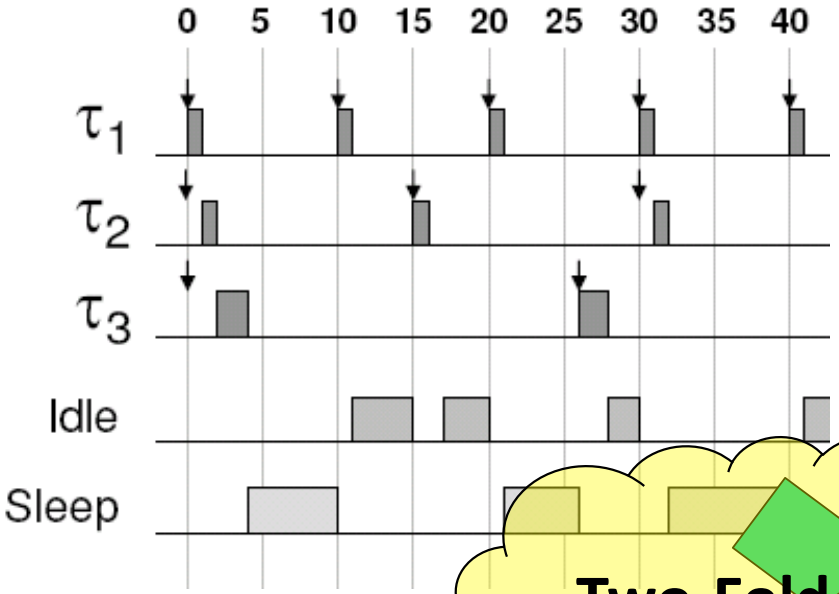
Rate Harmonized Scheduling[1]

- Pick a harmonizing period (<= shortest period)
- Release tasks only at the harmonizing interval



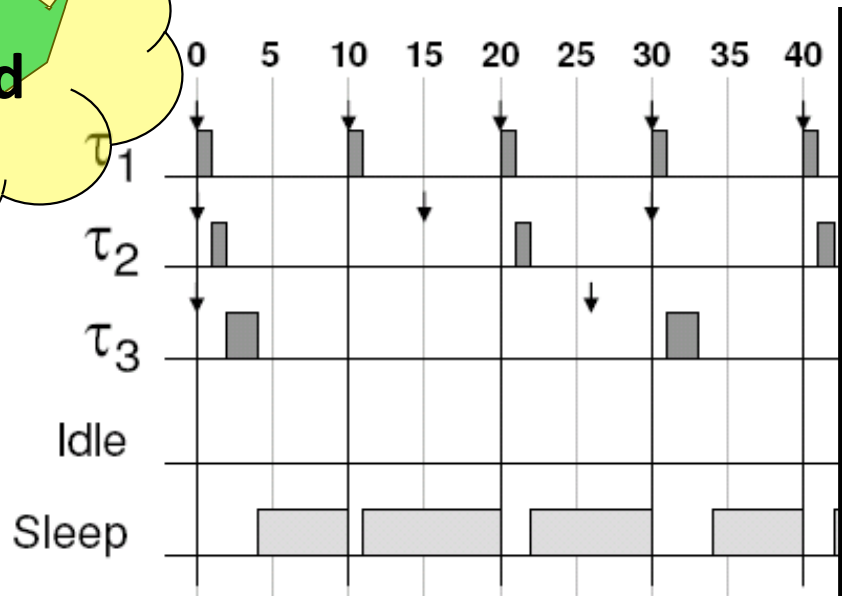
Anthony Rowe, Karthik Lakshmanan, Haifeng Zhu, Raj Rajkumar, "Rate-Harmonized Scheduling for Saving Energy", Proceedings of the 29th IEEE Real-Time Systems Symposium (RTSS), December 2008.

Transformation

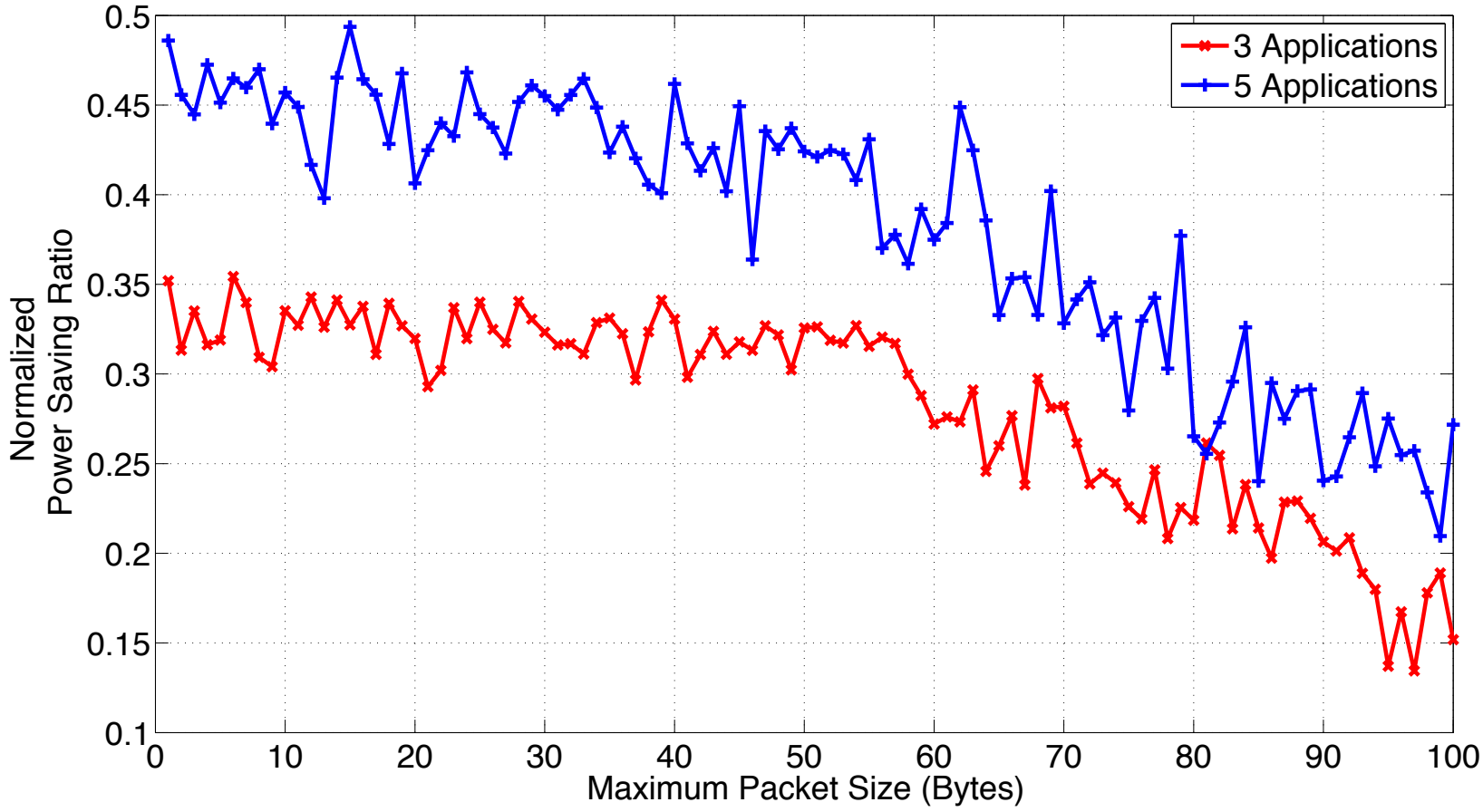


Two-Fold

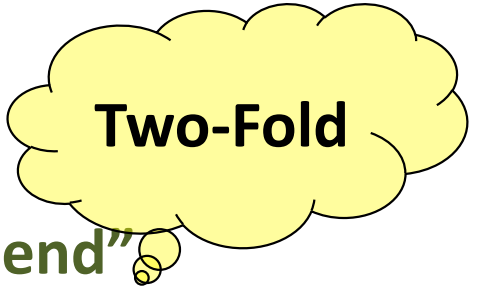
Apply RHS to
both processor
and Radio-Usage



Power Saving in Radio



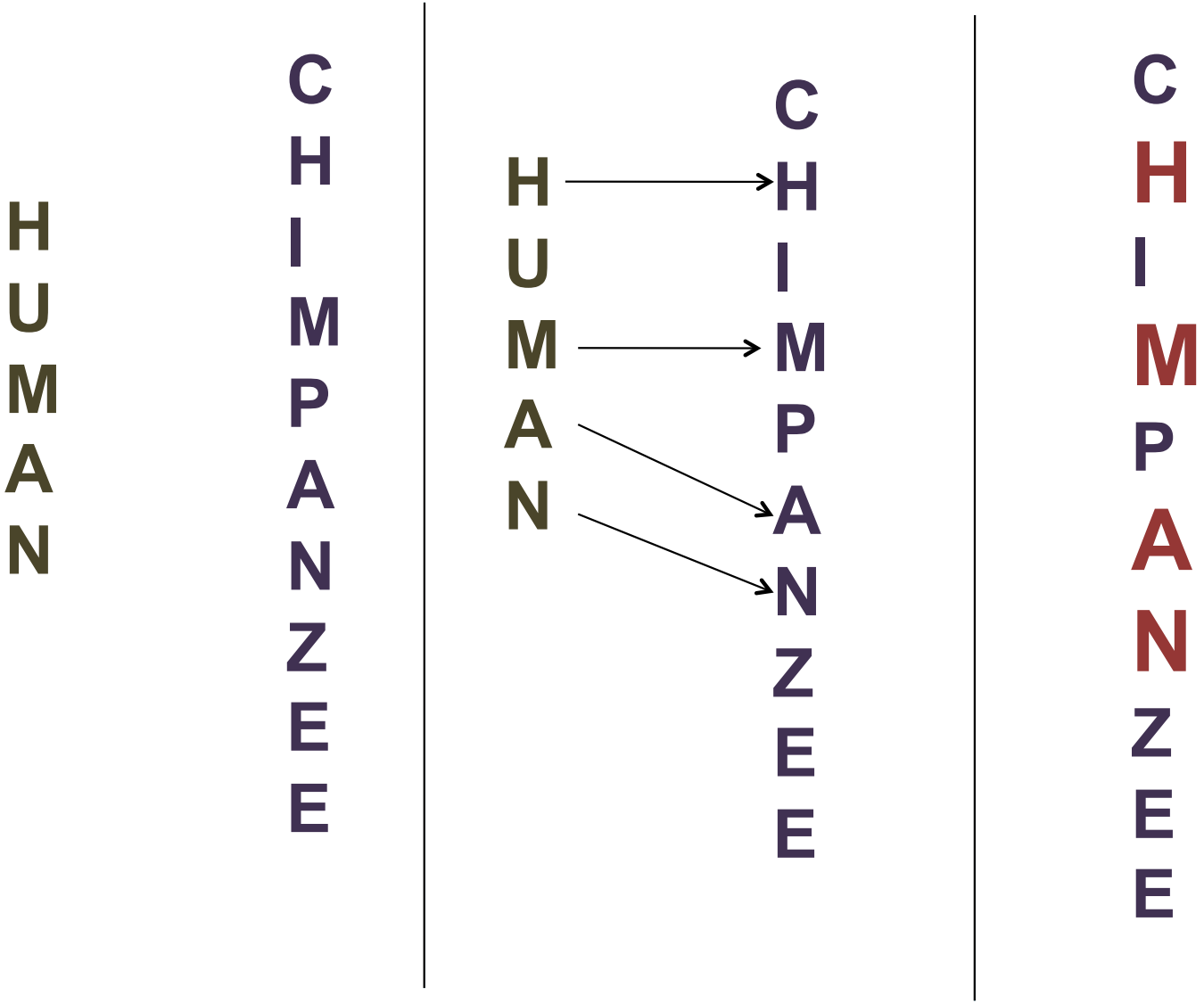
Not-So-Future Future-Work



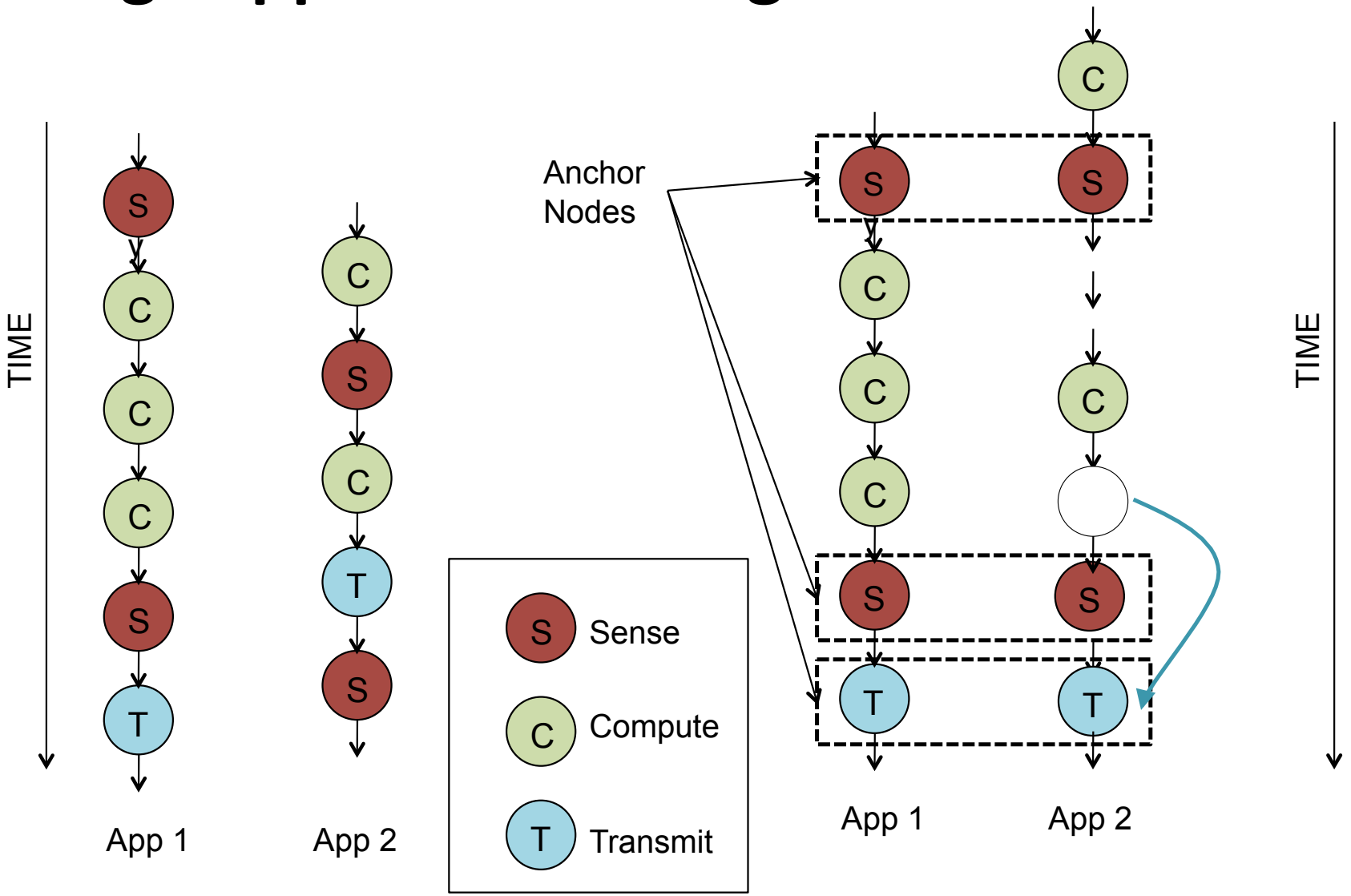
- Optimize multiple applications
- Reduce the redundancy in applications
- Applications centered around "Sense & Send"
- Remove the double work of sensing
- Sending already addressed

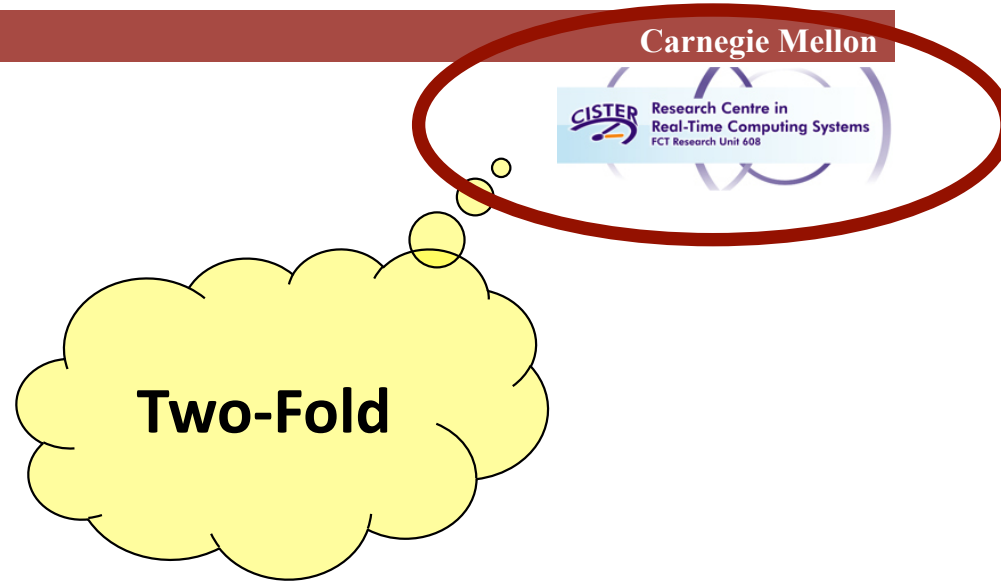
- Sample Light sensor only once
  - Share data among multiple applications

Longest Common Subsequence



Merge applications using LCS





I could go on with more slides

BUT VIK STOPS HERE ☺