Dandelion

Transparently programming phone-centered body sensor applications

Felix Xiaozhu Lin, Ahmad Rahmati, and Lin Zhong

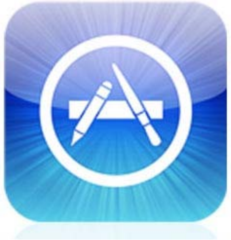
Rice Efficient Computing Group

Goal: transparently develop body sensor app

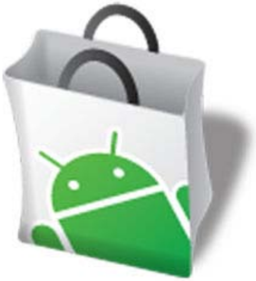
Challenge: the difficulty in programming sensors

Impact: ecosystem with numerous phone developers

Popular phone applications



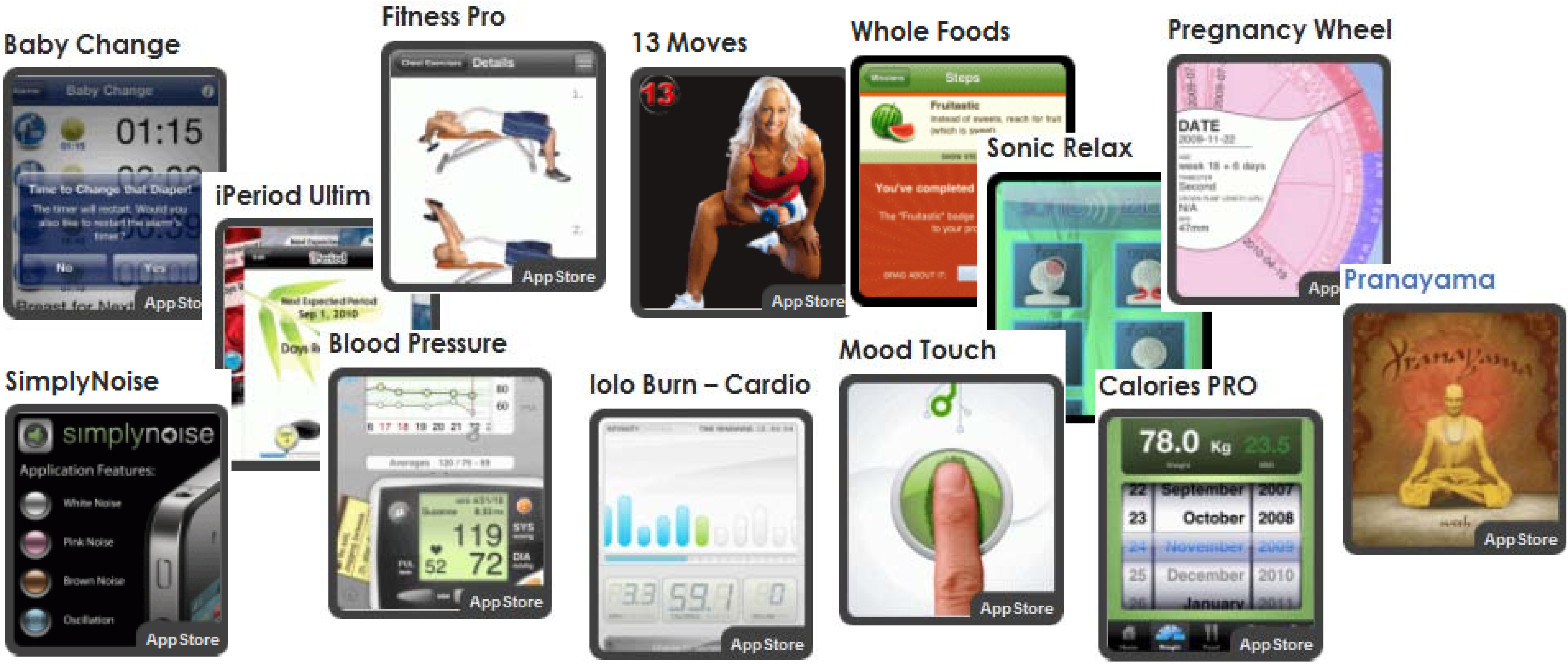
225,000



72,000

Source: Mobile Developer Economics 2010 and Beyond. Produced by VisionMobile. Sponsored by Telefonica Developer Communities. June 2010. Licensed under Creative Commons Attribution 3.0 License. Any use or remix of this work must retain this notice.

6000+ for health



http://articles.latimes.com/2010/jul/12/health/la-he-your-money-apps-20100712

Photo:Nokia Eco sensor

Phone + body sensors

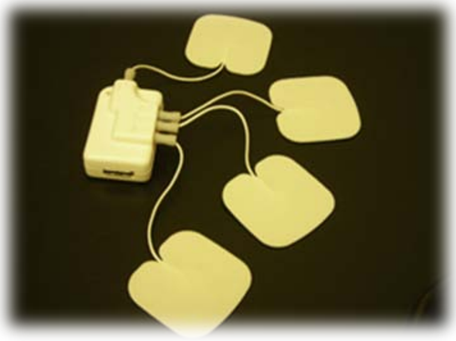
= *Innovative apps!*

How many apps utilize body sensors?

~%0

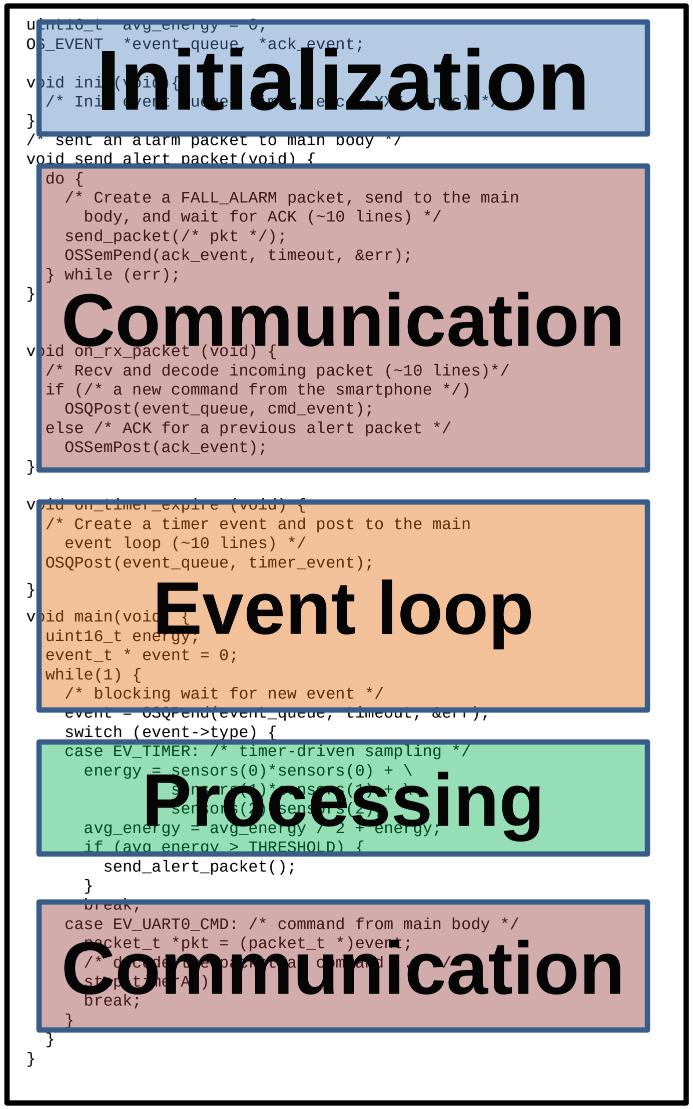
Why?

Body sensors **different** and **difficult** to program



Phone	Sensor
32-bit ARM processor	16-bit microcontroller
Java/C++/ObjC	Procedural, domain-specific
Linux/iOS	TinyOS/uCOS

Sensor style

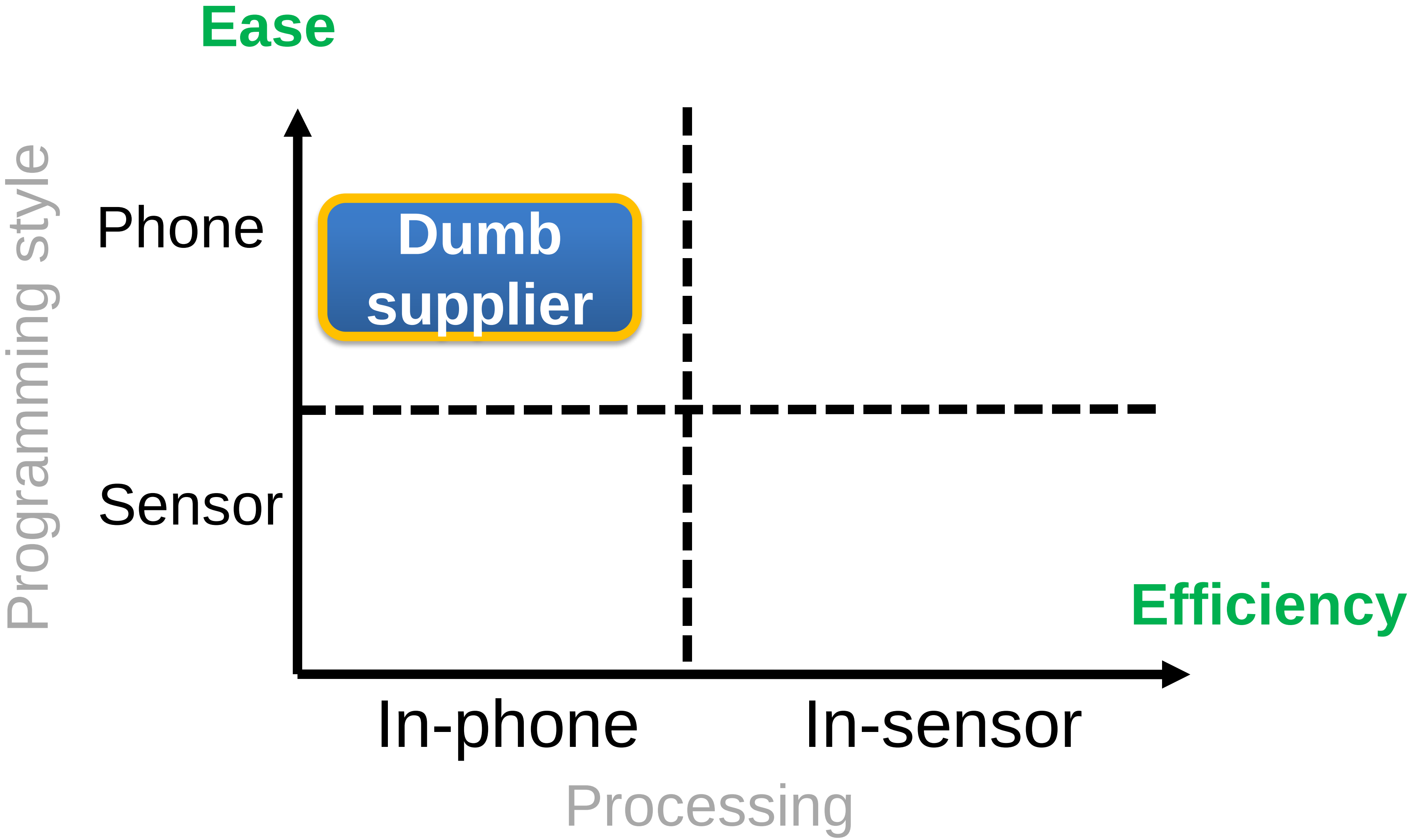


```
uint16_t  avg_energy = 0;
OS_EVENT  *event_queue, *ack_event;

void init(void){
 /* Init event queue, timer, etc (~XX lines) */
}
/* sent an alarm packet to main body */
void send_alert_packet(void) {
 do {
   /* Create a FALL_ALARM packet, send to the main
     body, and wait for ACK (~10 lines) */
   send_packet(/* pkt */);
   OSSemPend(ack_event, timeout, &err);
 } while (err);
}


void on_rx_packet (void) {
 /* Recv and decode incoming packet (~10 lines)*/
 if (/* a new command from the smartphone */)
   OSQPost(event_queue, cmd_event);
 else /* ACK for a previous alert packet */
   OSSemPost(ack_event);
}

void on_timer_expire (void) {
 /* Create a timer event and post to the main
   event loop (~10 lines) */
 OSQPost(event_queue, timer_event);

}
void main(void) {
 uint16_t energy;
 event_t * event = 0;
 while(1) {
   /* blocking wait for new event */
   event = OSQPend(event_queue, timeout, &err);
   switch (event->type) {
   case EV_TIMER: /* timer-driven sampling */
     energy = sensors(0)*sensors(0) + \
          sensors(1)*sensors(1) + \
          sensors(2)*sensors(2);
     avg_energy = avg_energy / 2 + energy;
     if (avg_energy > THRESHOLD) {
       send_alert_packet();
     }
     break;
   case EV_UART0_CMD: /* command from main body */
     packet_t *pkt = (packet_t *)event;
     /* decode the packet as command ... */
     stop_timerA()
     break;
   }
  }
}
```

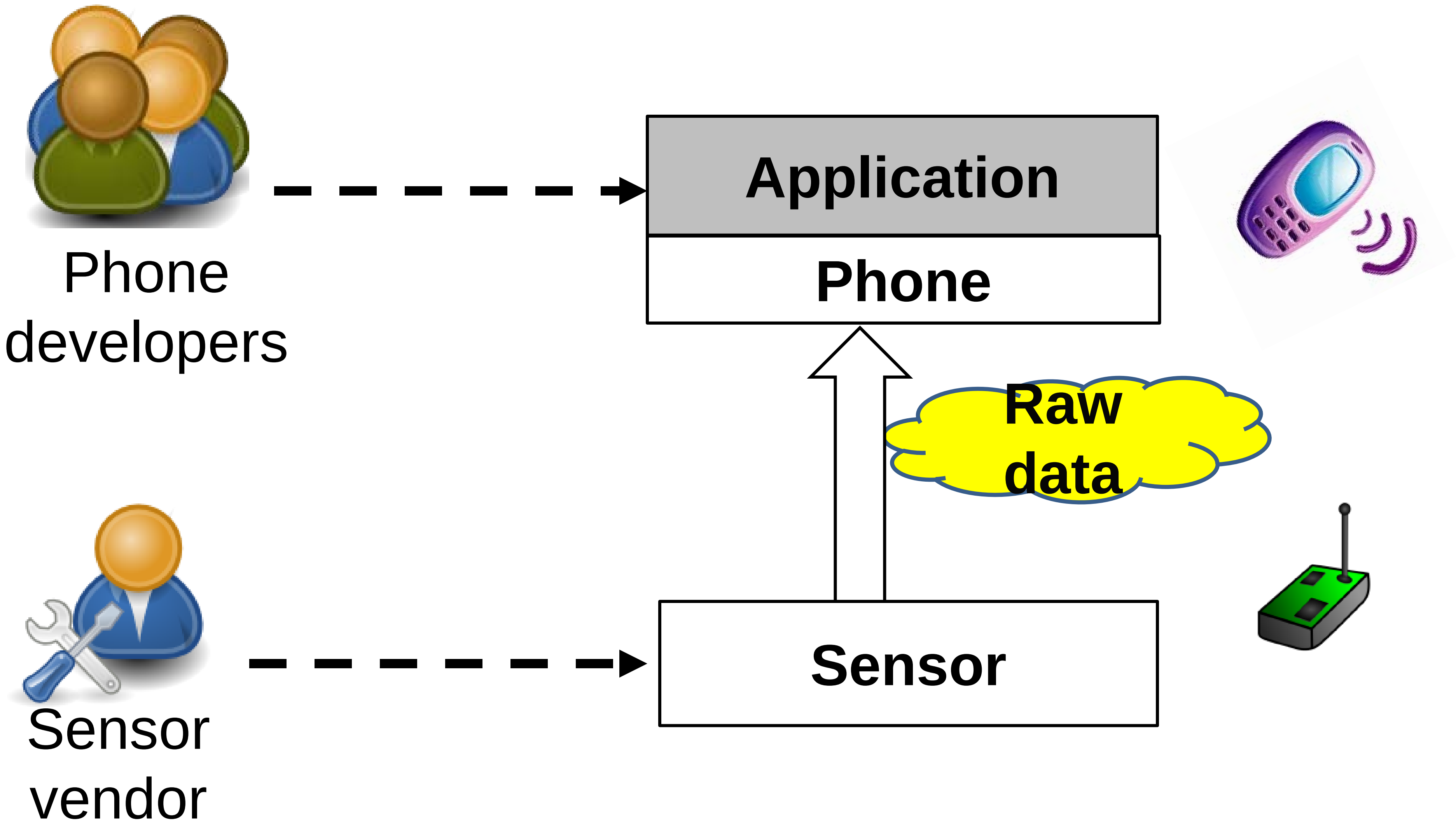
≠

Phone style

```
class MyListener : SensorListener {
public:
  OnCreate() { ... };
  OnDestroy() { ... };
  OnNewData(sensor_id, data) { ...
  };
private:
  // private states as variables
}
```


Ease
Programming style
Phone
Sensor
Dumb supplier
Efficiency
In-phone
In-sensor
Processing

Sensor as *dumb* data supplier



Phone developers

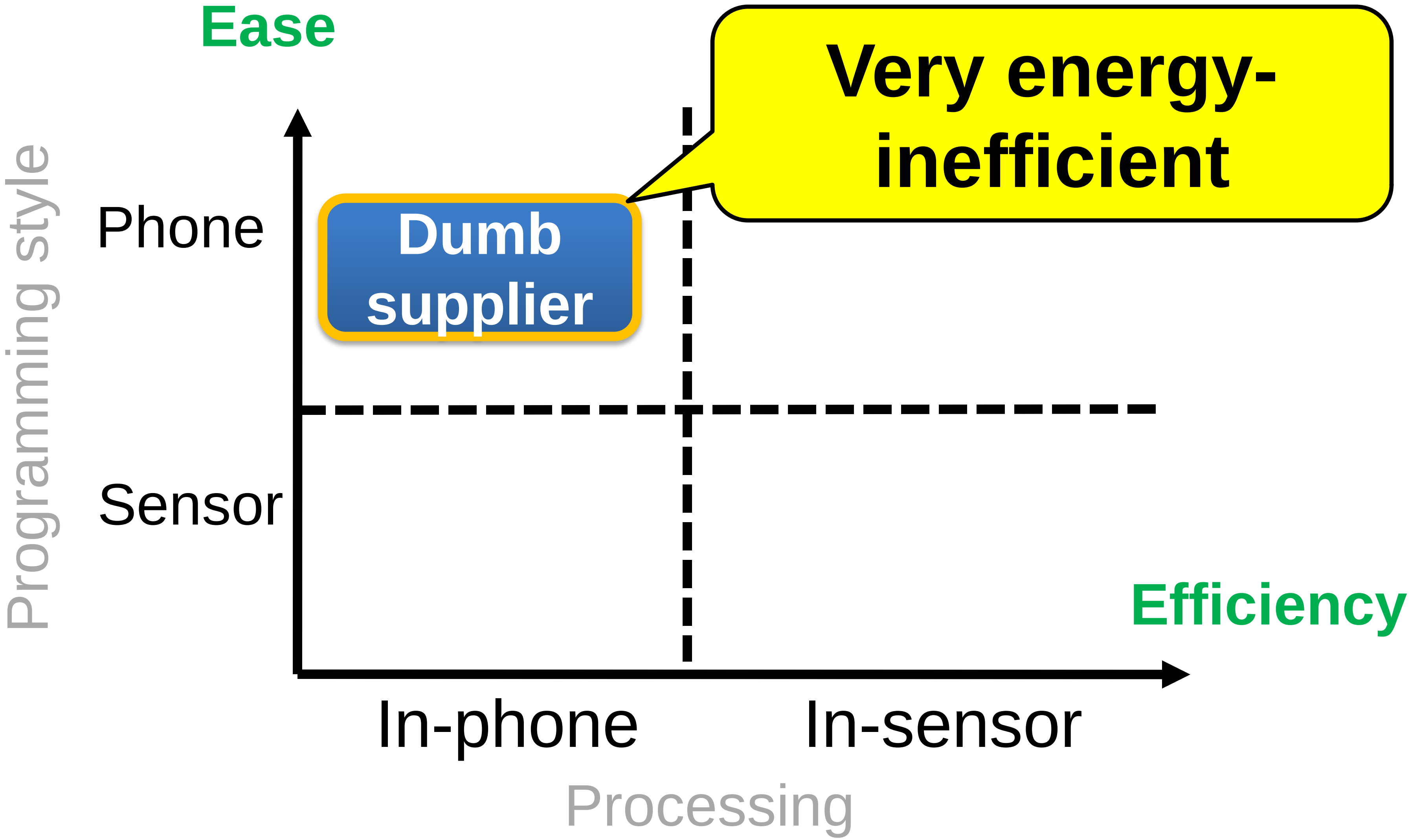
Application

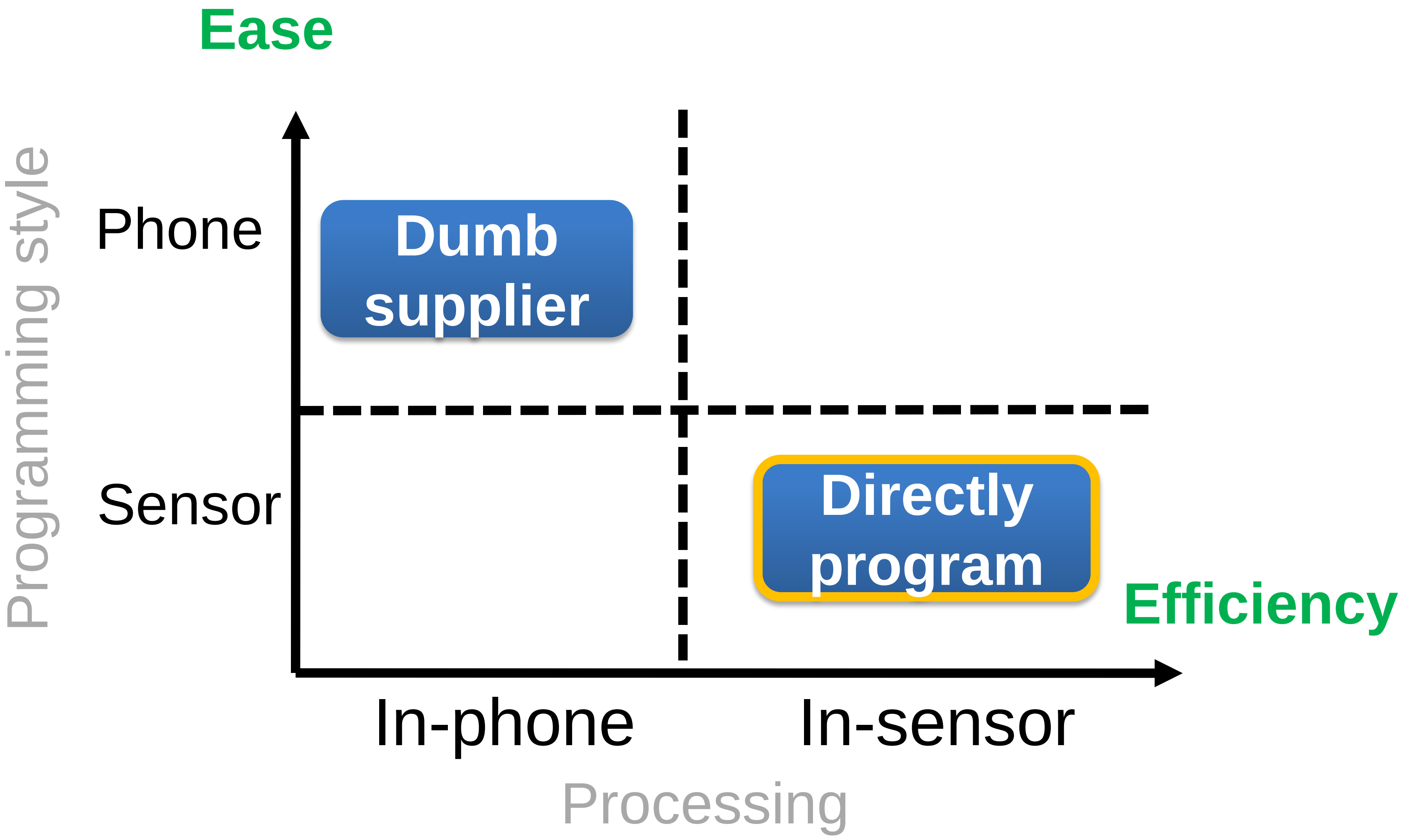
Phone

Raw data

Sensor

Sensor vendor


Ease
Very energy-inefficient
Programming style
Phone
Dumb supplier
Sensor
Efficiency
In-phone
In-sensor
Processing



Ease

Programming style

Phone

Sensor

Dumb supplier

Directly program

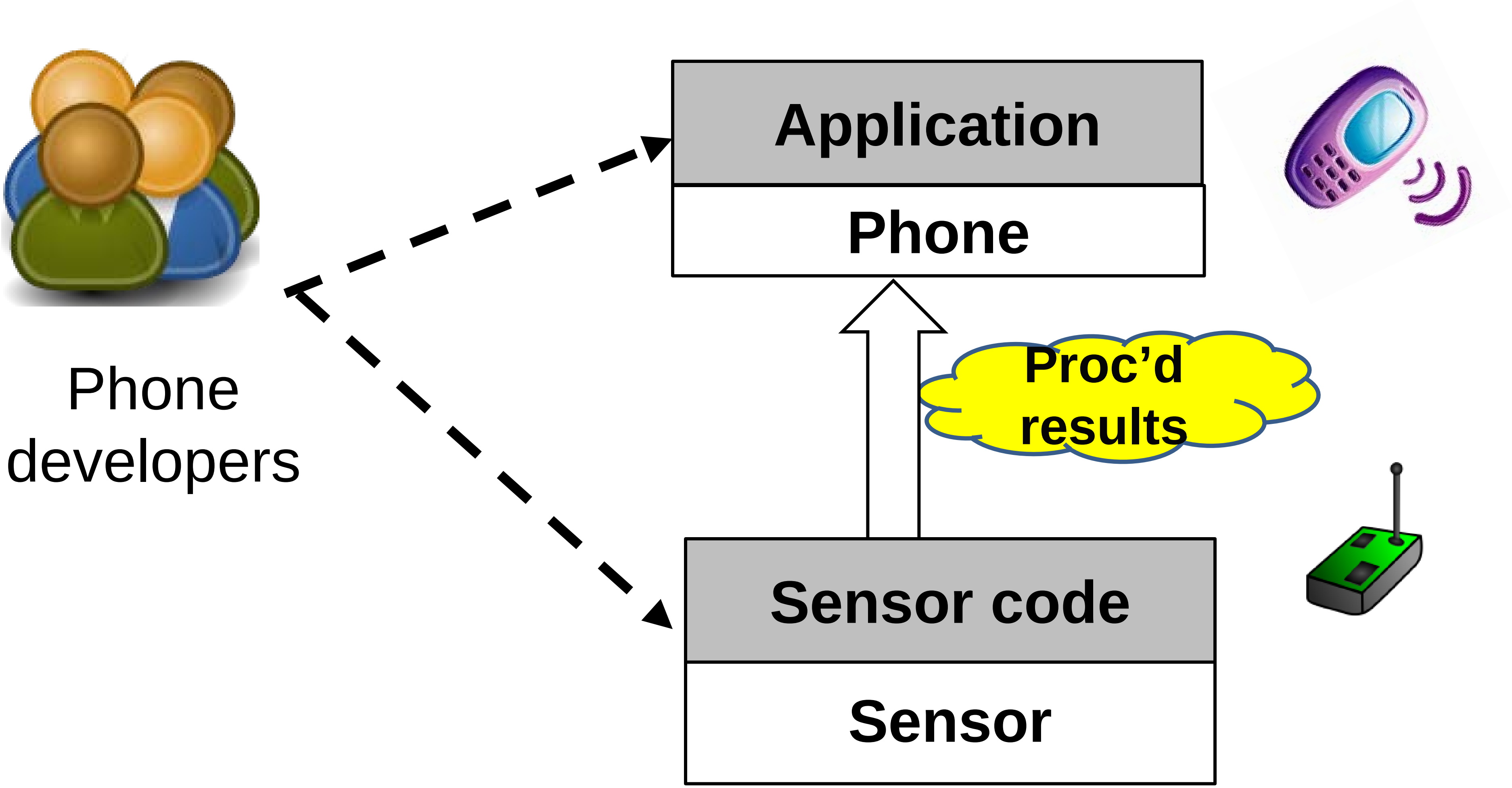
Efficiency

In-phone

In-sensor

Processing

Directly program the sensor



Phone developers

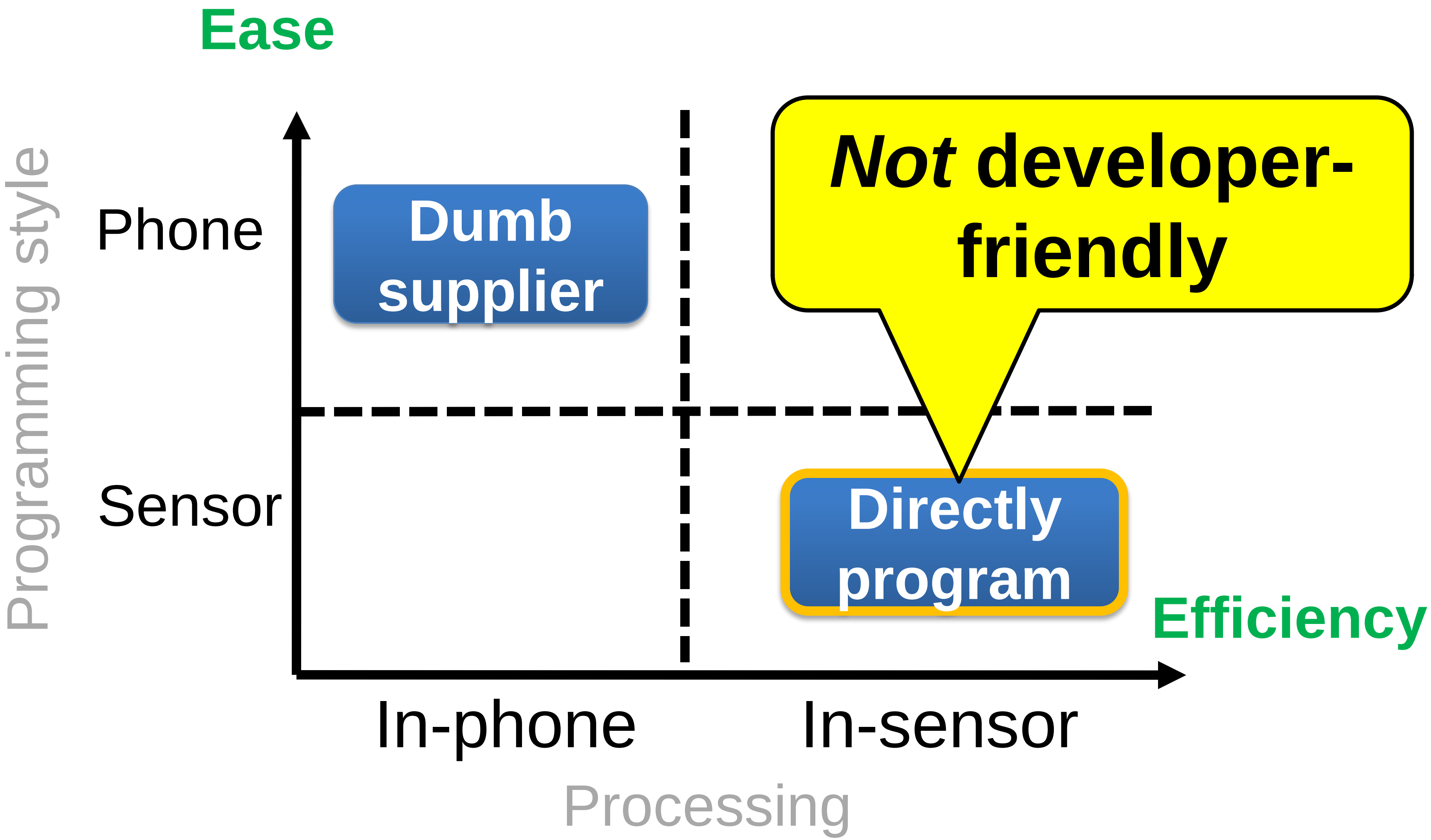
Application

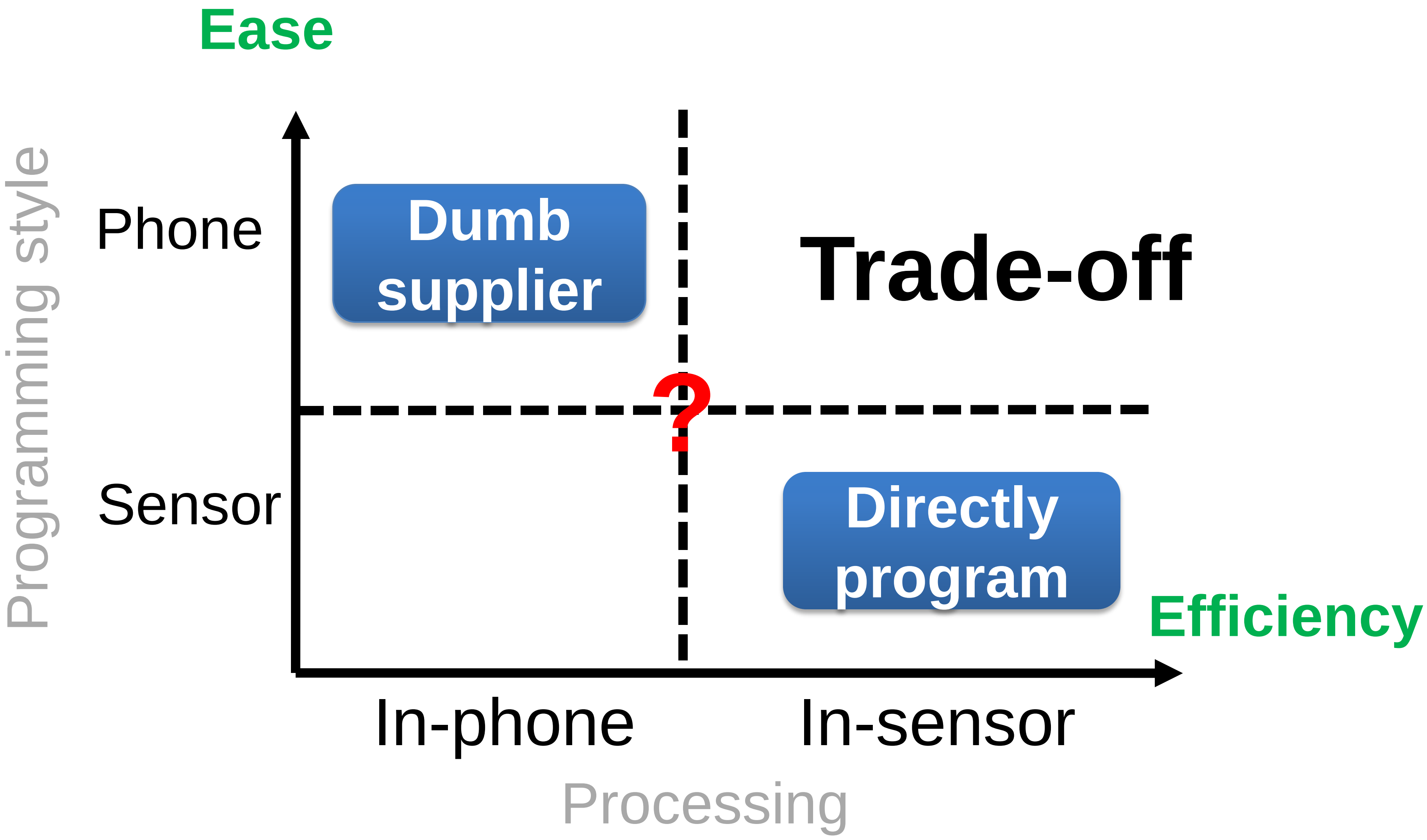
Phone

Proc’d results

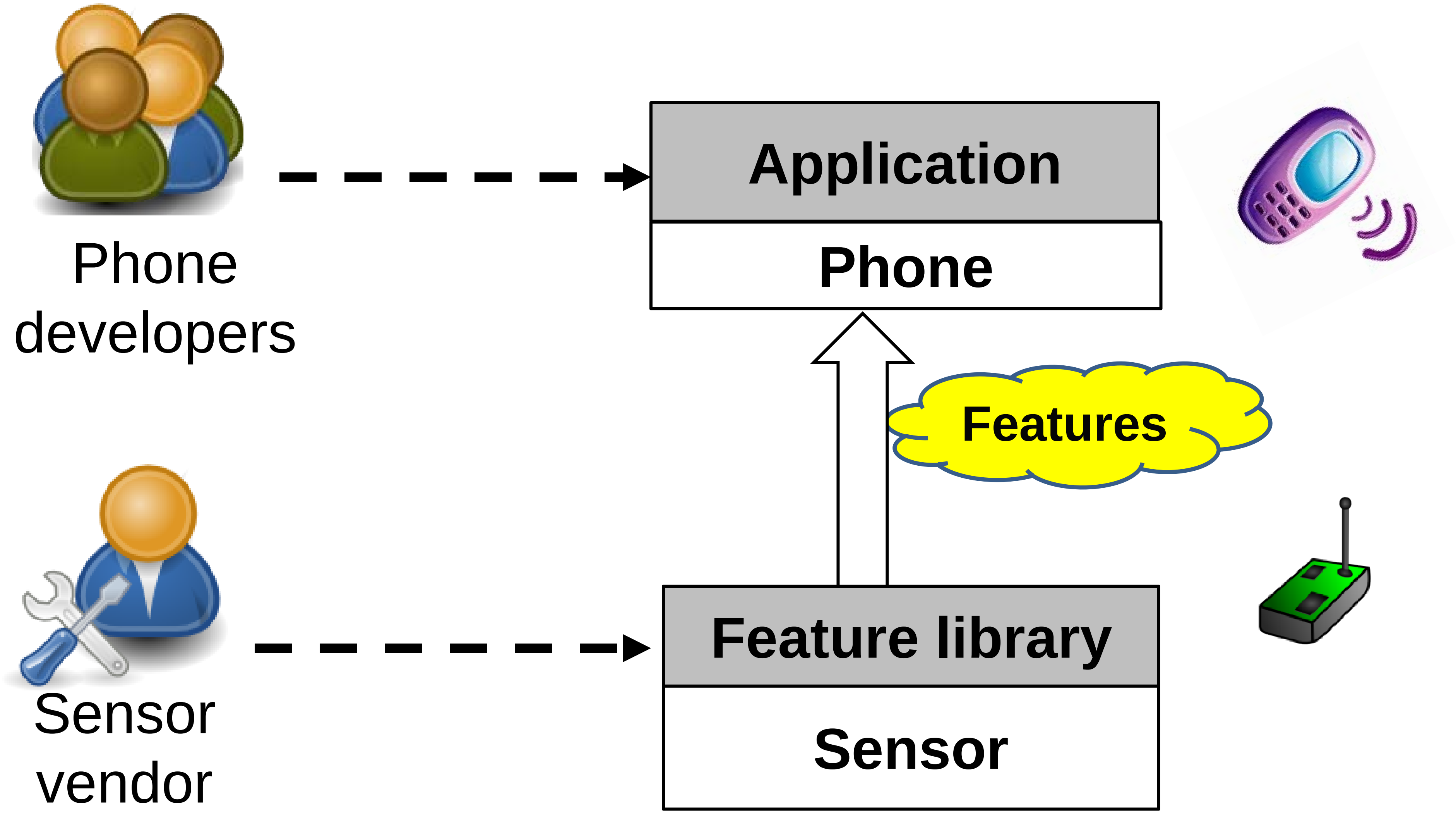
Sensor code

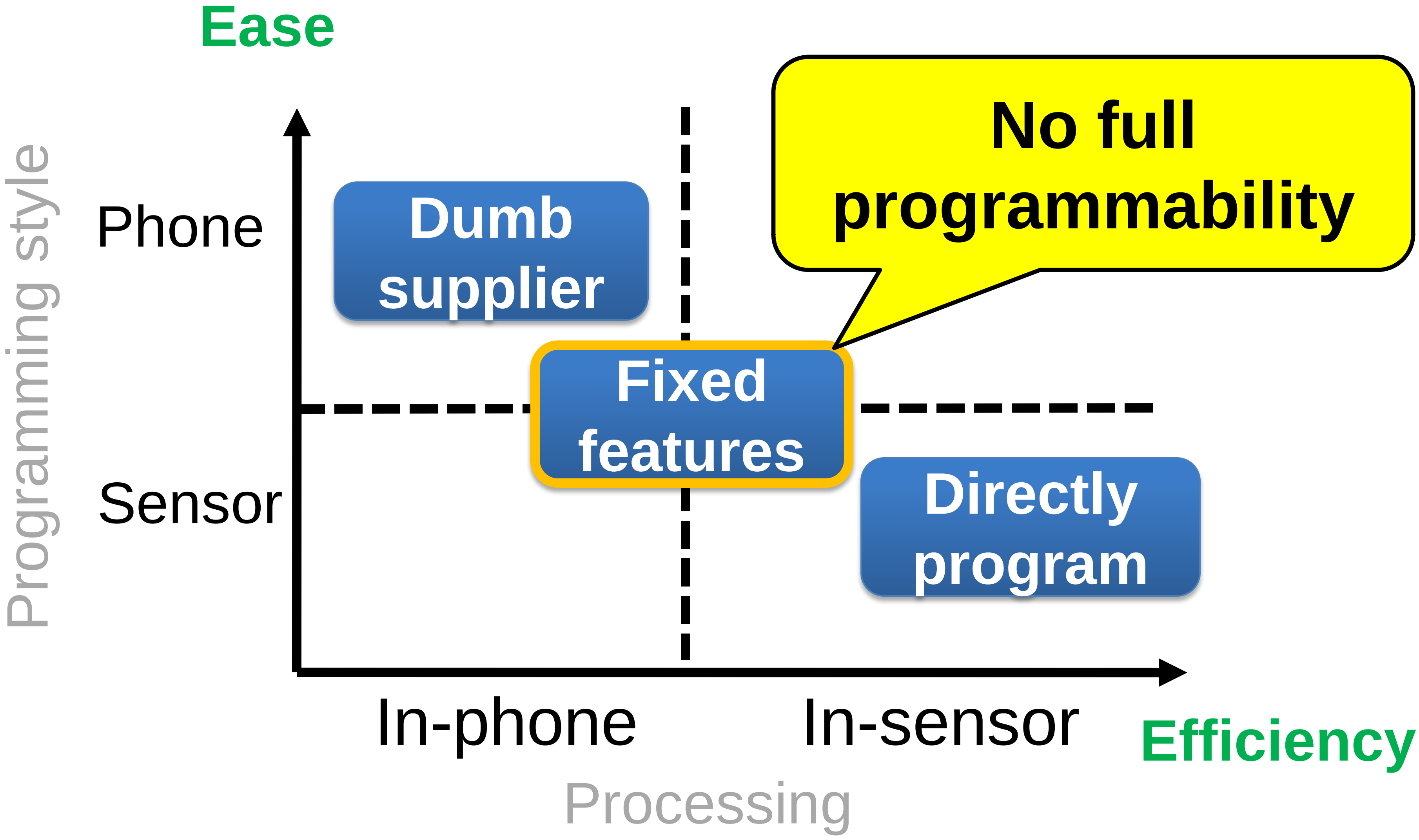
Sensor


Ease
Programming style
Phone
Sensor
Dumb supplier
Not developer-friendly
Directly program
Efficiency
In-phone
In-sensor
Processing


Ease
Programming style
Phone
Sensor
Dumb supplier
Trade-off
?
Directly program
Efficiency
In-phone
In-sensor
Processing

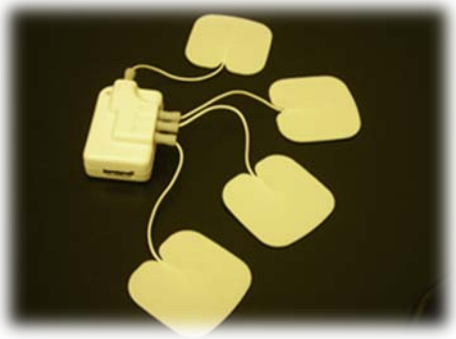
Fixed features




Ease
No full programmability
Programming style
Phone
Dumb supplier
Fixed features
Sensor
Directly program
In-phone
In-sensor
Efficiency
Processing

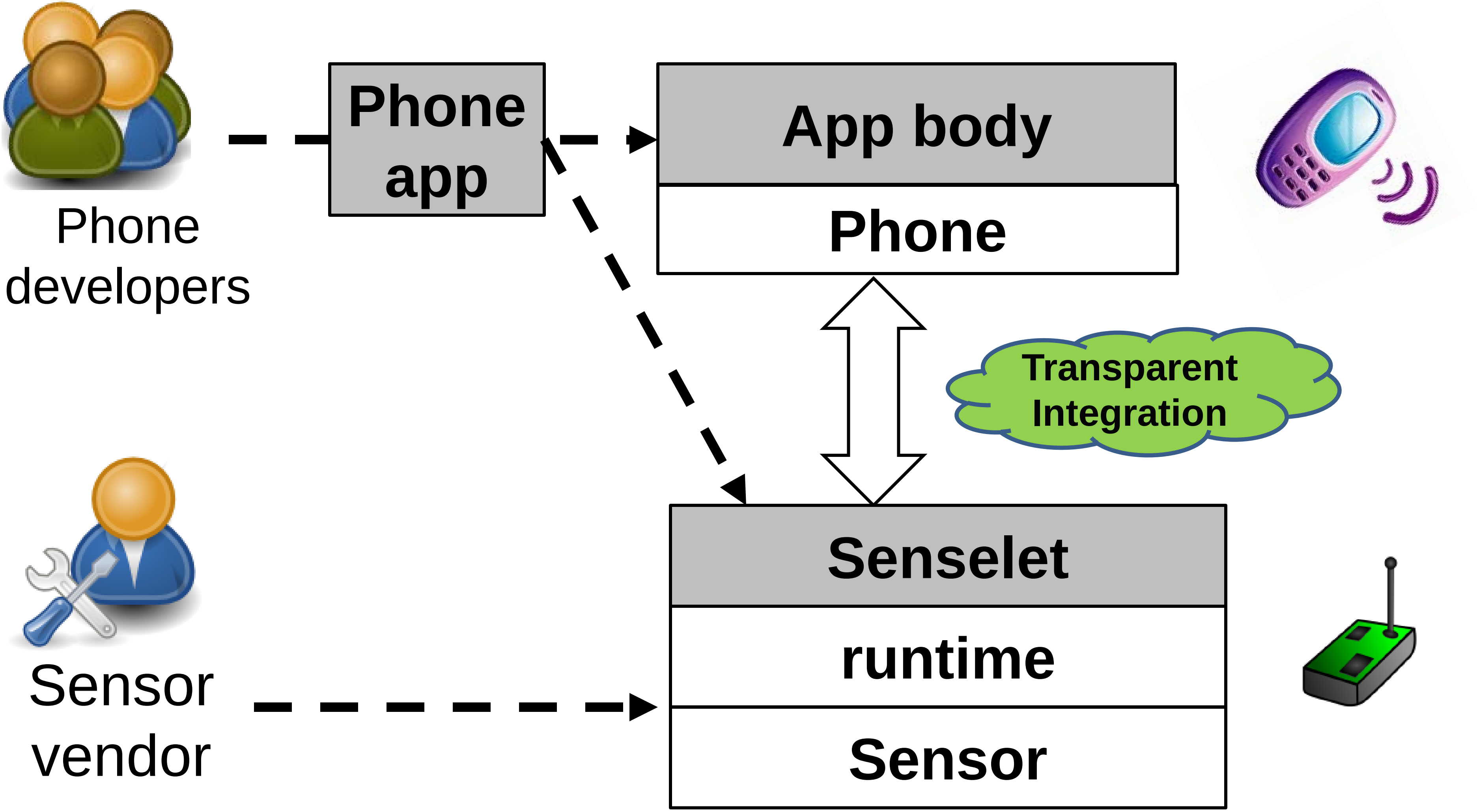
Dandelion

Transparently develop body sensor apps



Phone	Sensor
32-bit ARM processor	16-bit microcontroller
Java/C++/ObjC	Procedural domain-specific
Linux/iOS	TinyOS/uCOS

Transparency



Phone developers

Phone app

App body

Phone

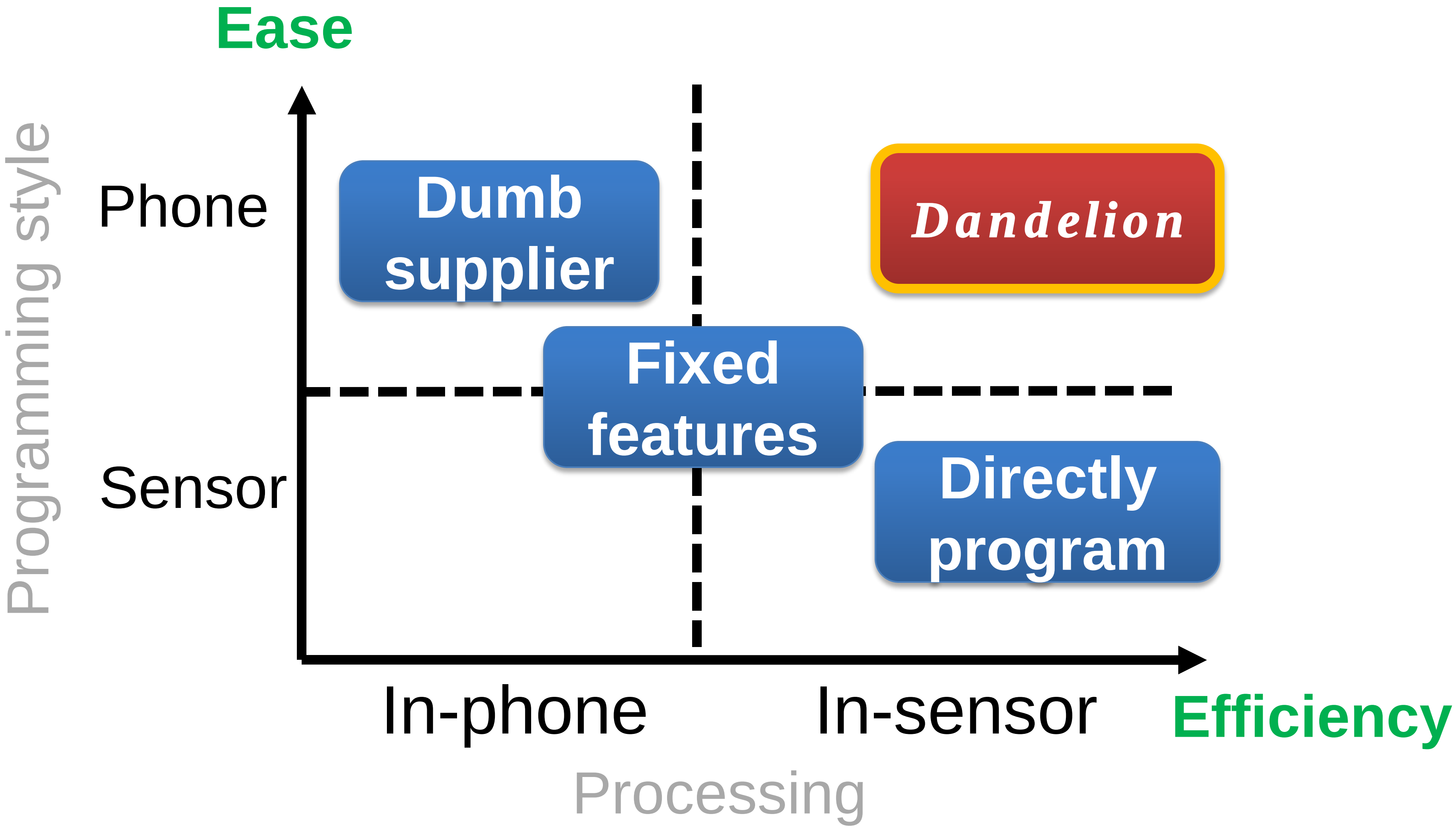
Transparent Integration

Senselet

runtime

Sensor

Sensor vendor



Ease

Programming style

Phone

Sensor

Dumb supplier

Dandelion

Fixed features

Directly program

In-phone

In-sensor

Efficiency

Processing

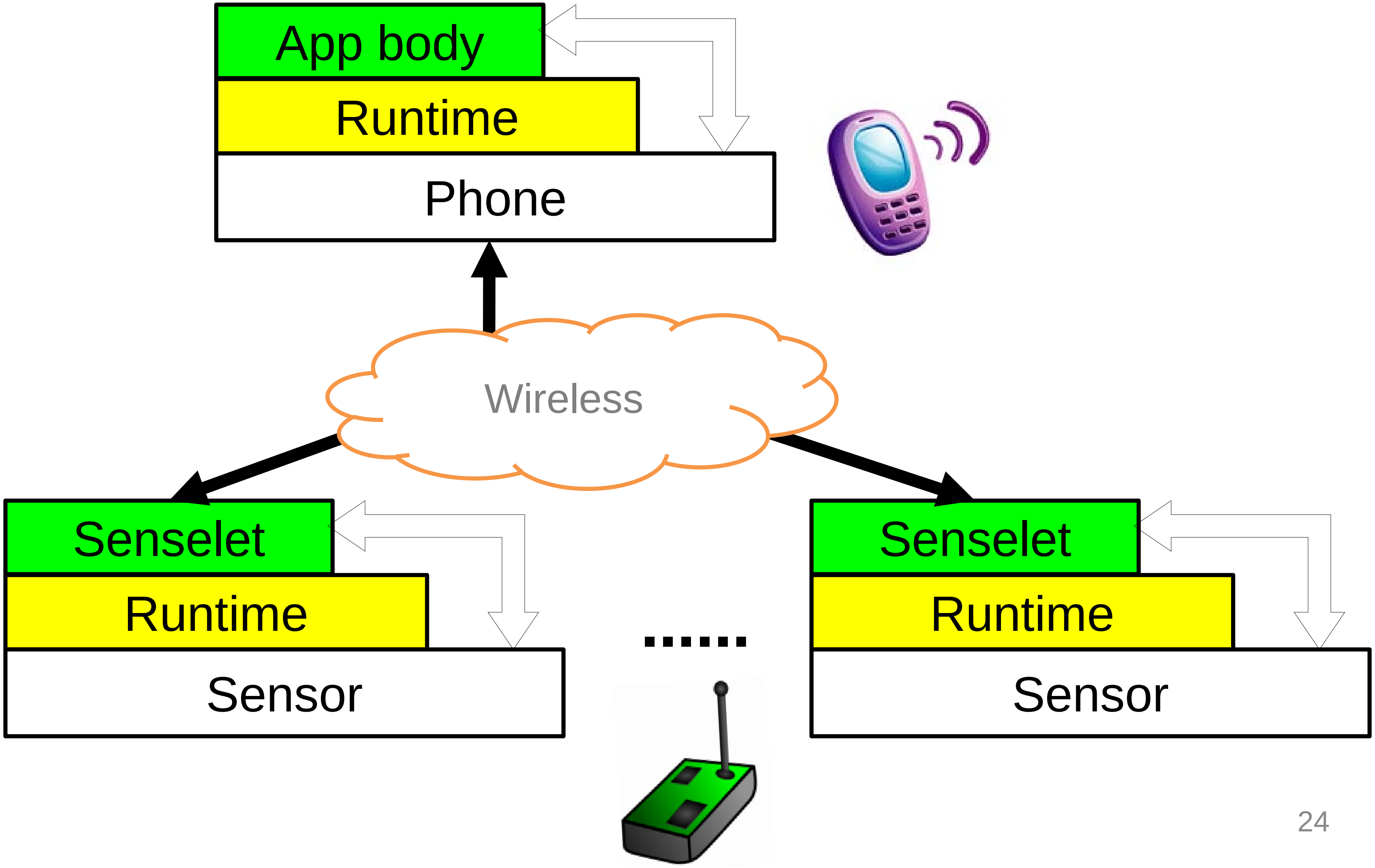
System Design

The runtime system

Programming abstraction

Programming support

The runtime system



App body

Runtime

Phone

Wireless

Senselet

Runtime

Sensor

......

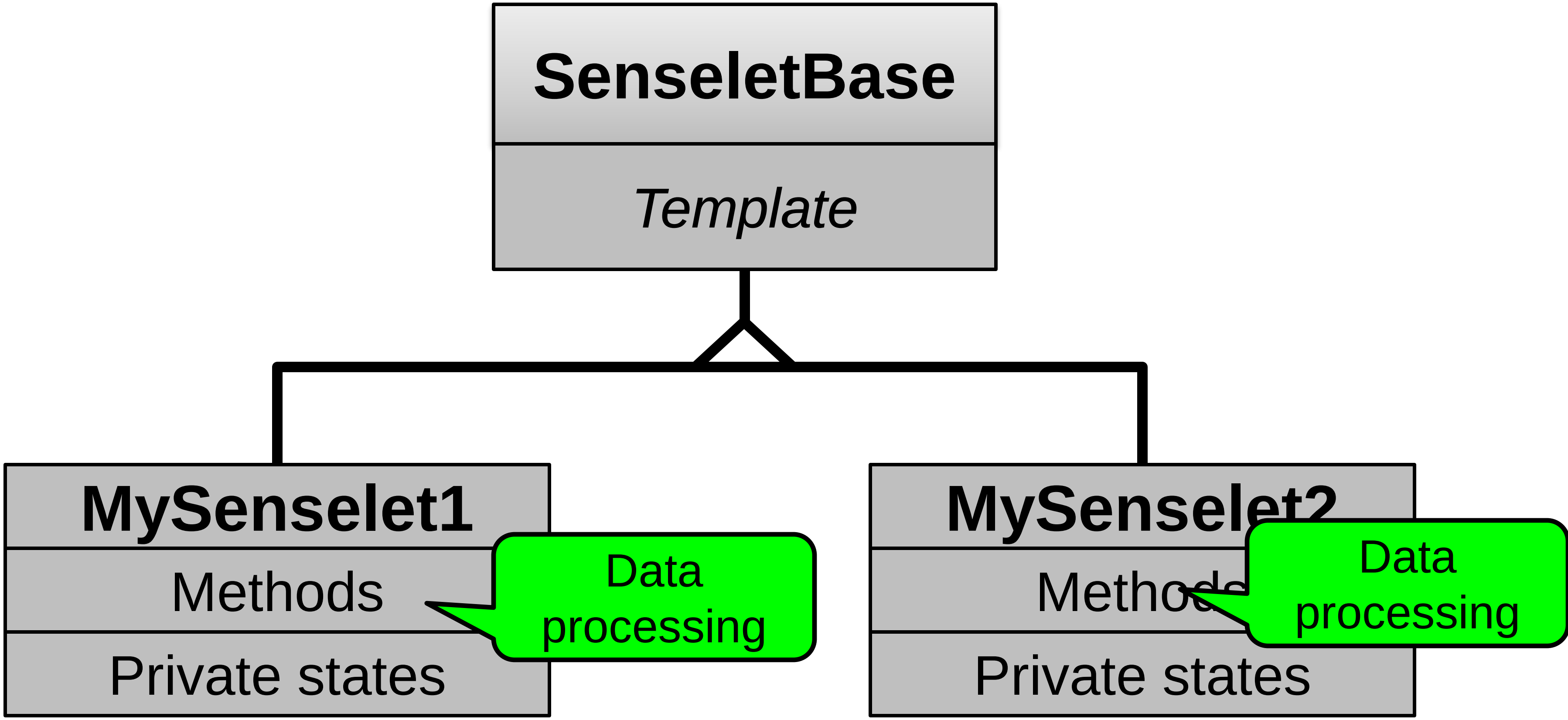
Senselet

Runtime

Sensor

Senselet

Programming abstraction



SenseletBase

Template

MySenselet1

Methods

Private states

Data processing

MySenselet2

Methods

Private states

Data processing

Programming Support

Platform services

Remote method invocation

Compile & deploy

Three platform services

Essential + widely supported

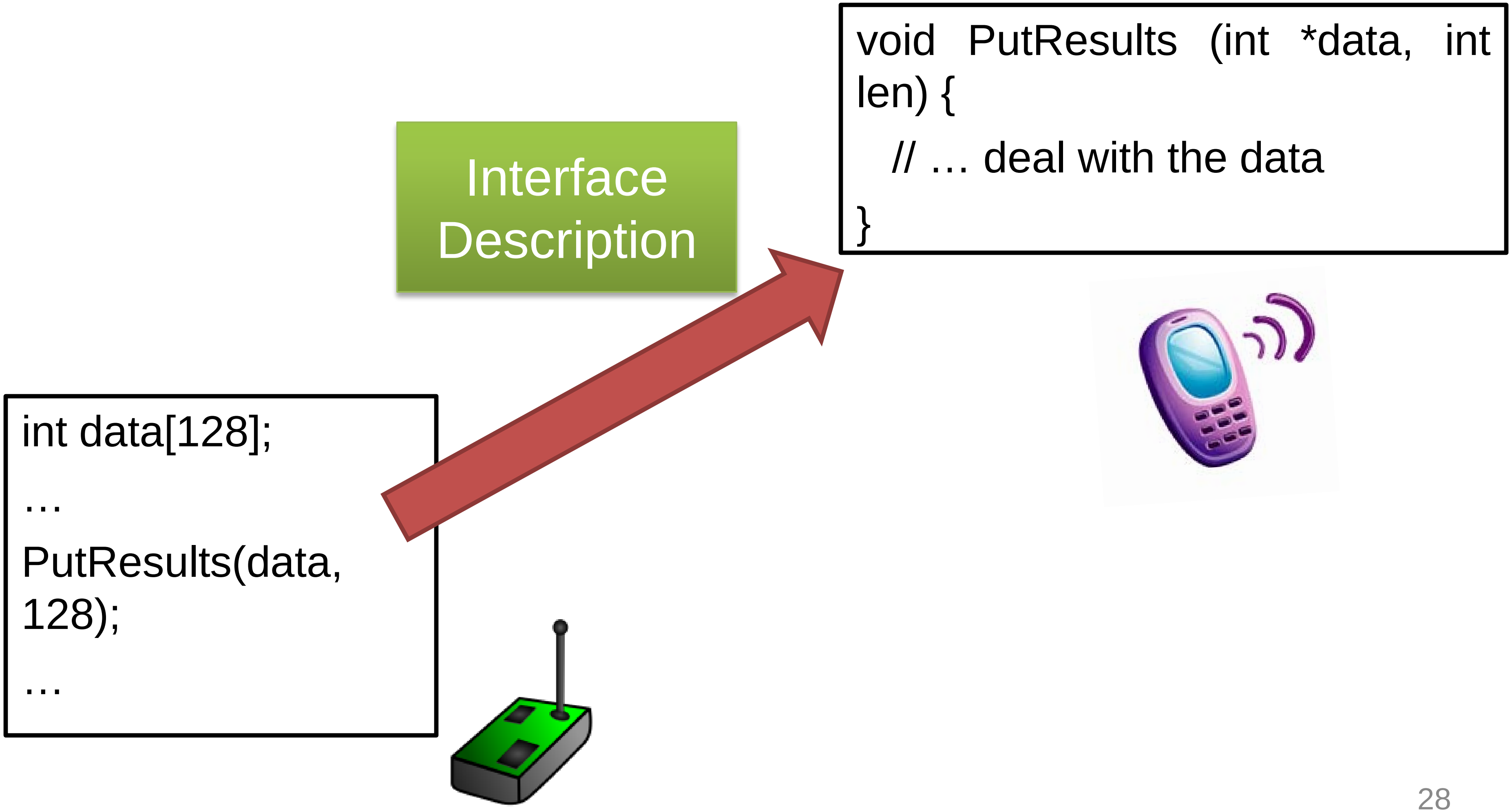
Data acquisition

Timer

Dynamic memory

Remote method

Function calls across a senselet and the app body

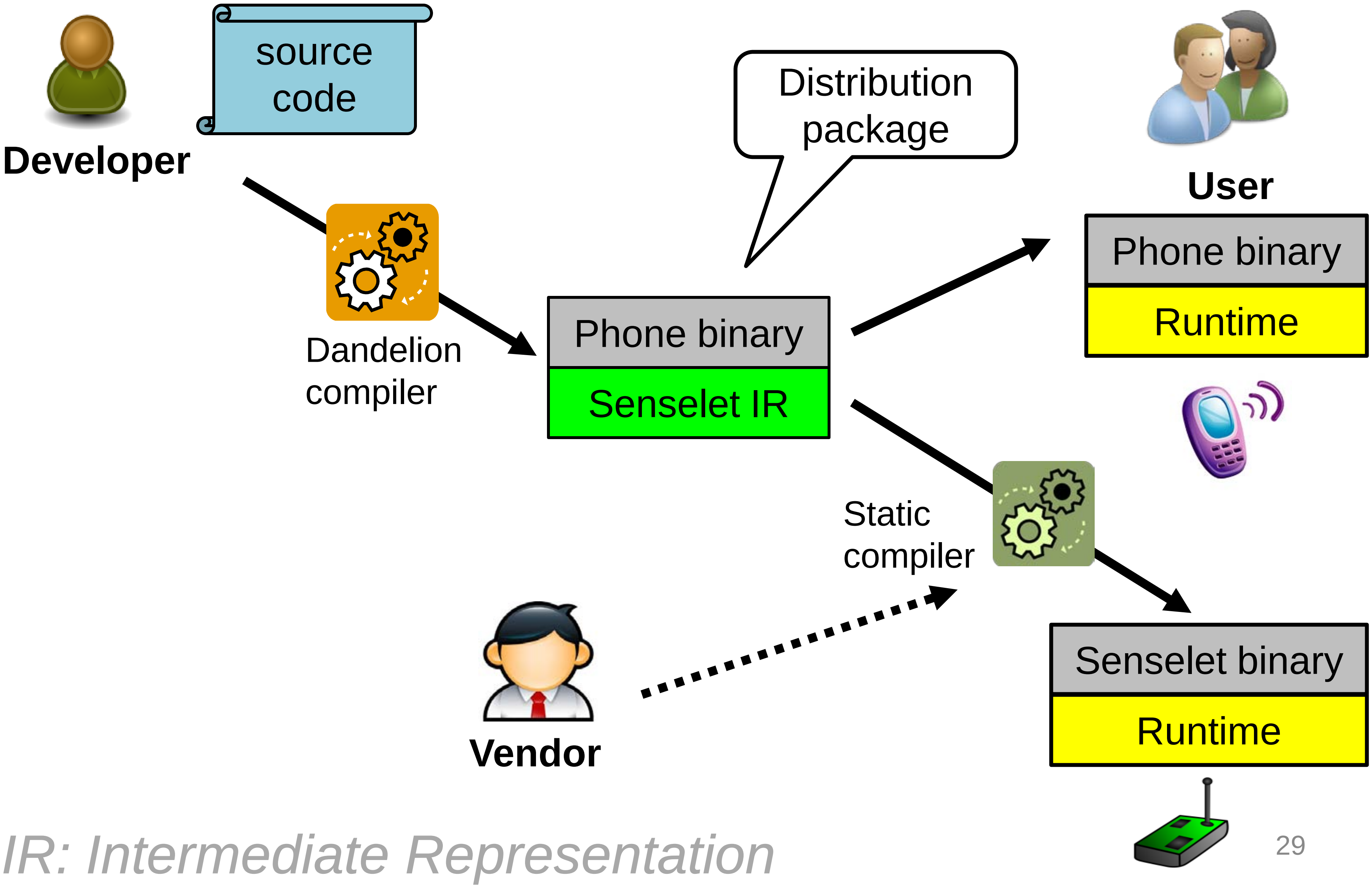


Interface Description

```
void PutResults (int *data, int
len) {
   // … deal with the data
}
```

```
int data[128];
…
PutResults(data,
128);
…
```

Target-independent compilation



Developer

source code

Dandelion compiler

Phone binary

Senselet IR

Distribution package

User

Phone binary

Runtime

Static compiler

Vendor

Senselet binary

Runtime

IR: Intermediate Representation

Example: Fall detector

```
class SenseletFall : public SenseletBase {
public:
  void OnCreate() {_avg_energy = 0; RegisterSensorData(ACCEL, 50);};

  void OnData (uint8_t *readings, uint16_t len) {
    uint16_t energy = readings[0]*readings[0] + readings[1]*readings[1] + \
                      readings[2]*readings[2];
    //do a simple low-pass filtering
    _avg_energy = _avg_energy / 2 + energy / 2;

    // detect fall accident with the filtered energy
    if (_avg_energy > THRESHOLD) { theMainBody.FallAlert(); }
  }
  void OnDestroy () {UnRegisterSensorData(ACCEL);}

private:
  uint16_t _avg_energy;
};
```

Platform service

Periodic processing

Remote method

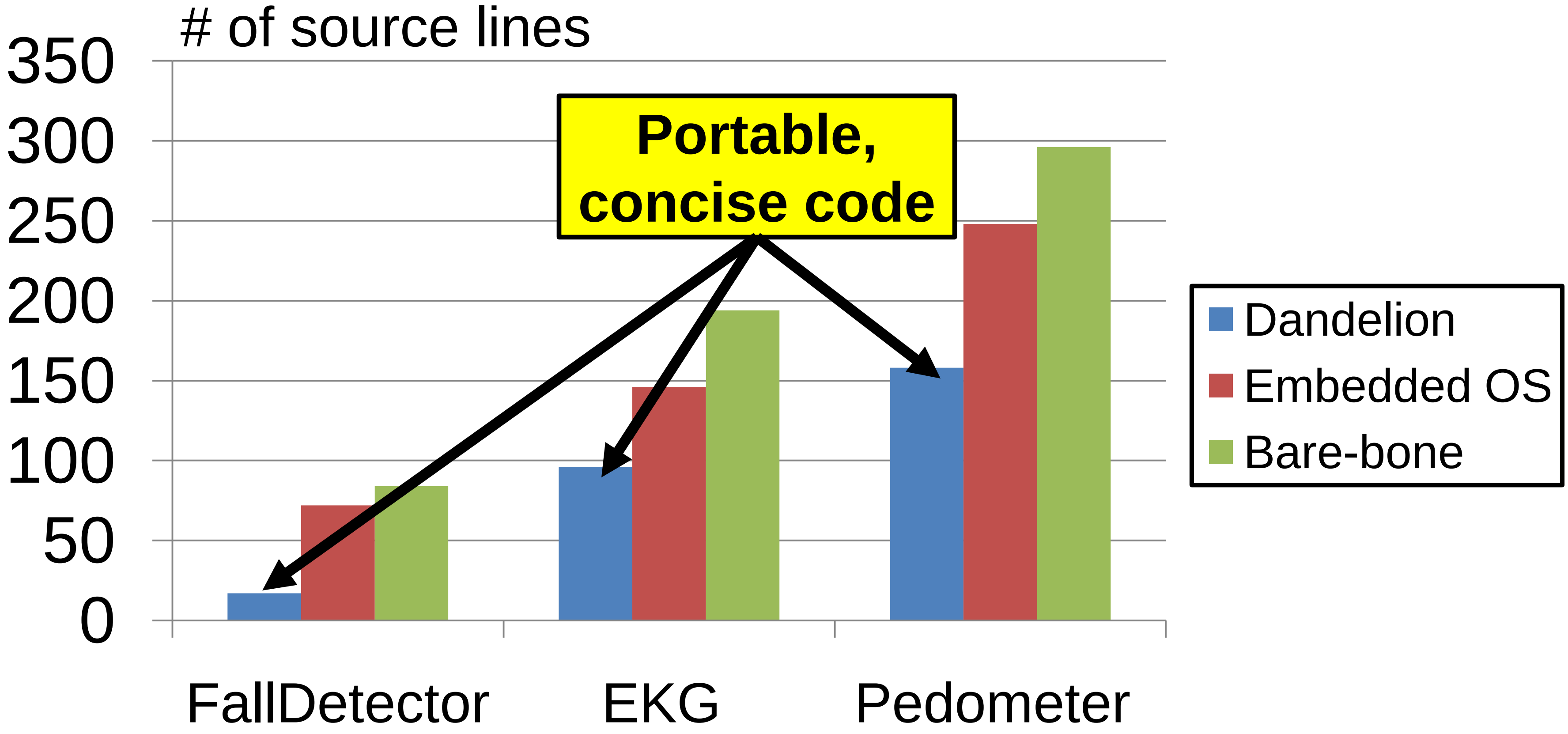
Implementation

Platform + Dandelion + body sensor apps

Nokia N900
Rice Orbit body sensor
LLVM compiler infrastructure

Source code comparison

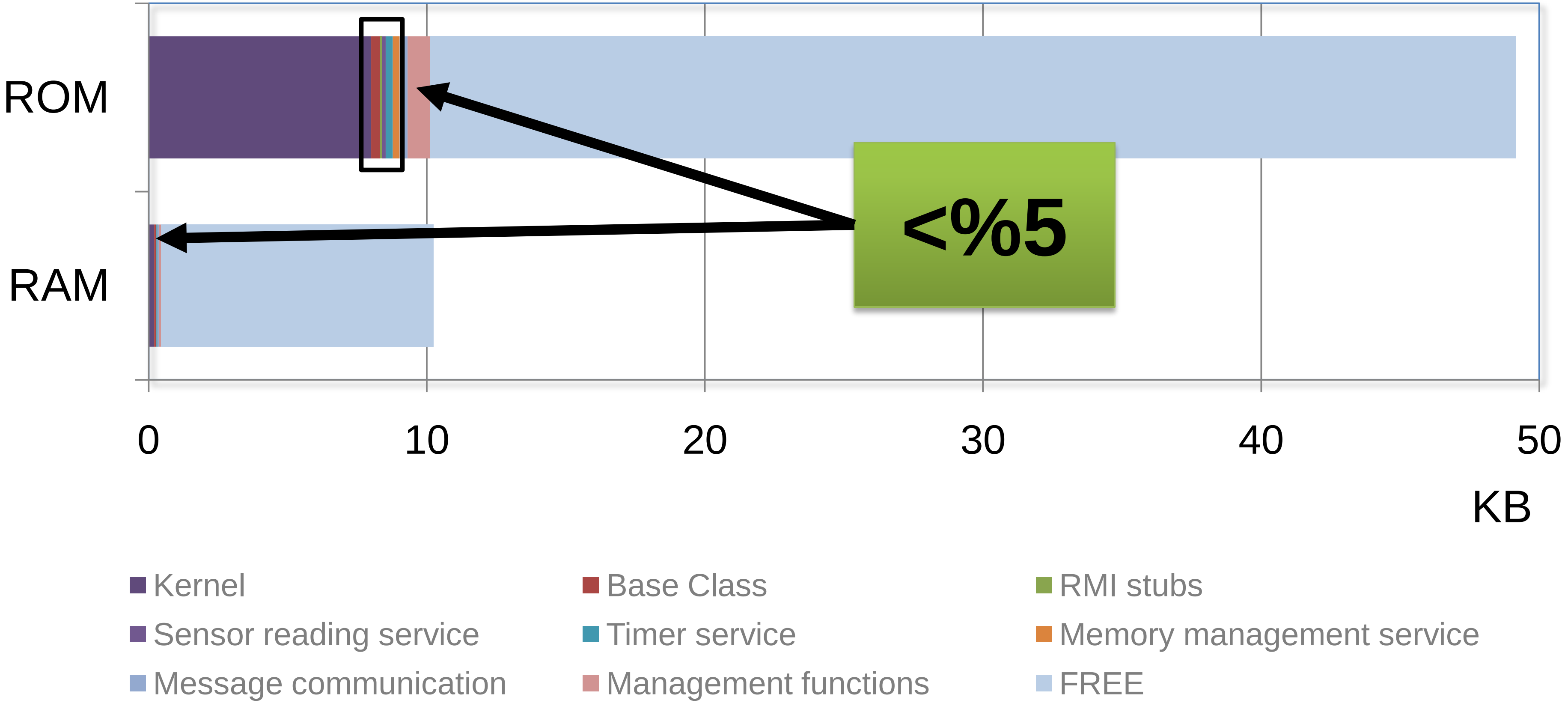
Three apps, each with three implementations



Portable, concise code

Memory overhead

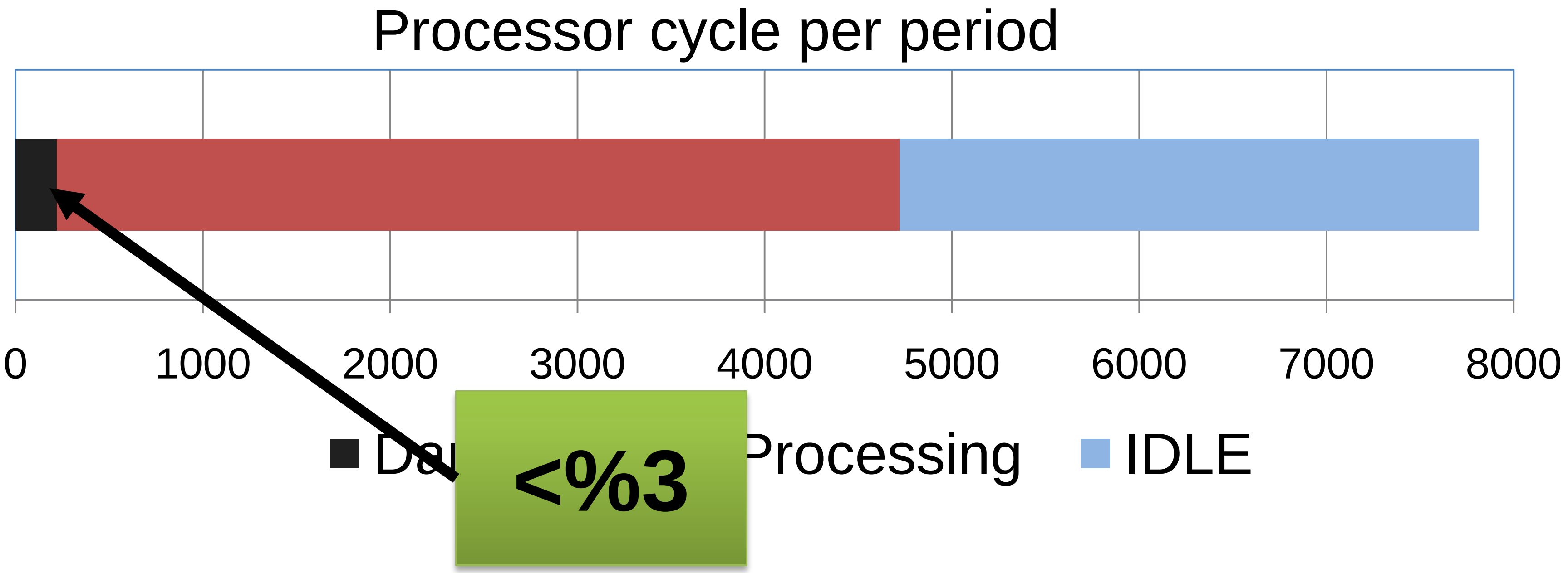
Measured on MSP430



<%5

Execution overhead

Measured on MSP430

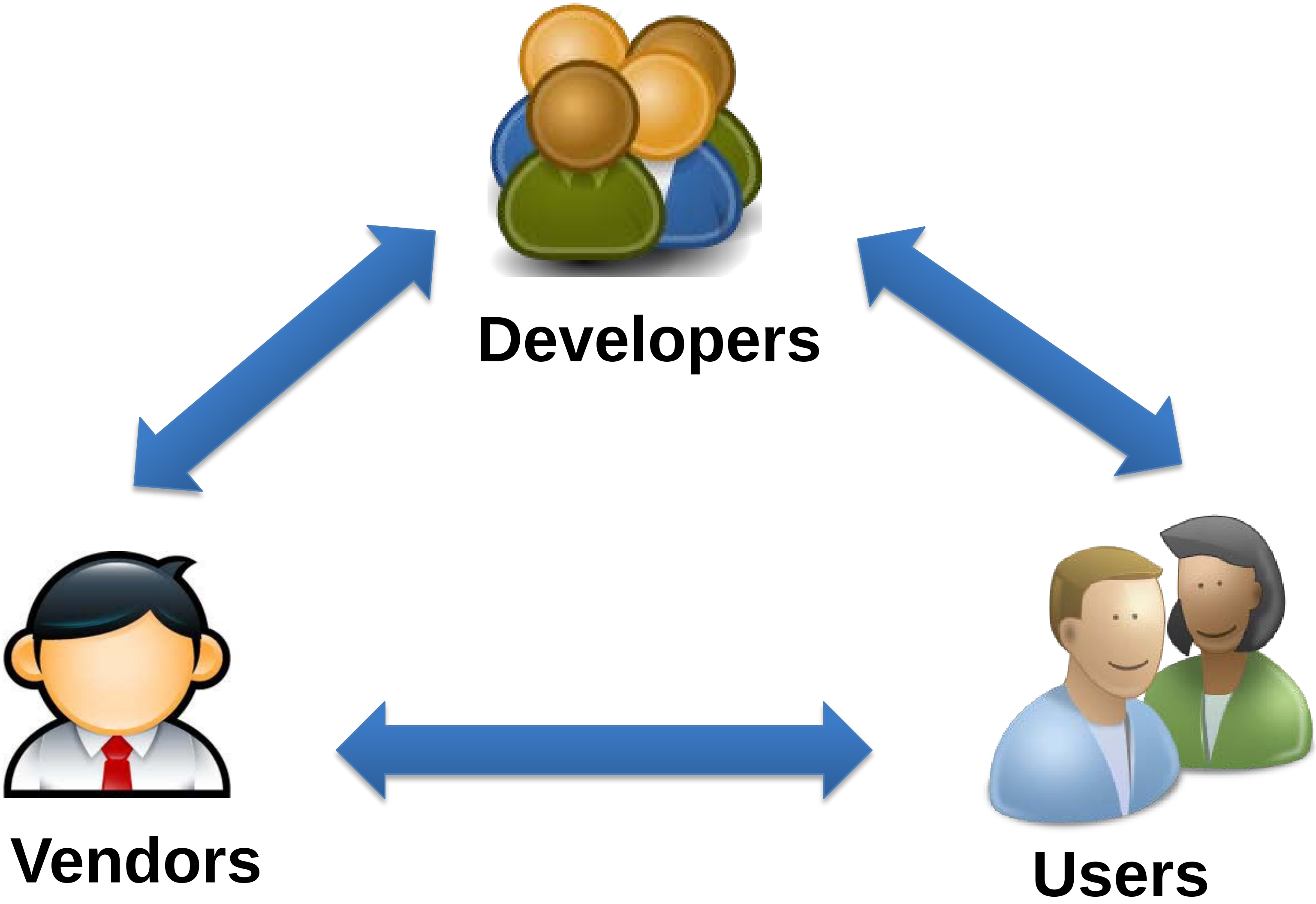


<%3

Conclusions

- Dandelion enables transparently programming body sensor apps
- Dandelion incurs very small overhead

The Ecosystem



Developers

Vendors

Users

http://www.cs.rice.edu/~xl6/dandelion/

THANKS!