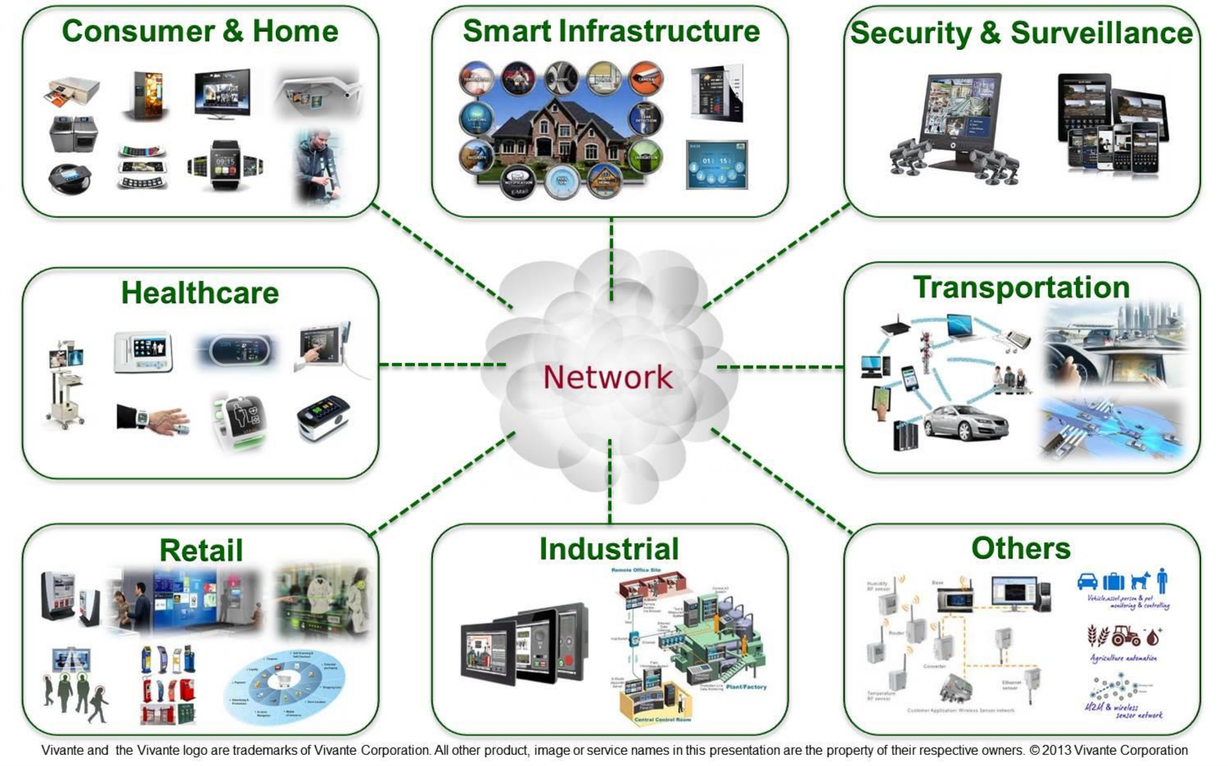
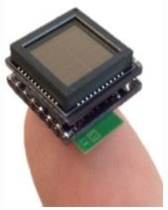
Torpor: A Power-Aware HW Scheduler for Energy Harvesting IoT SoCs

02.07.2018

Internet-of-Things SoCs



~20 cm^2

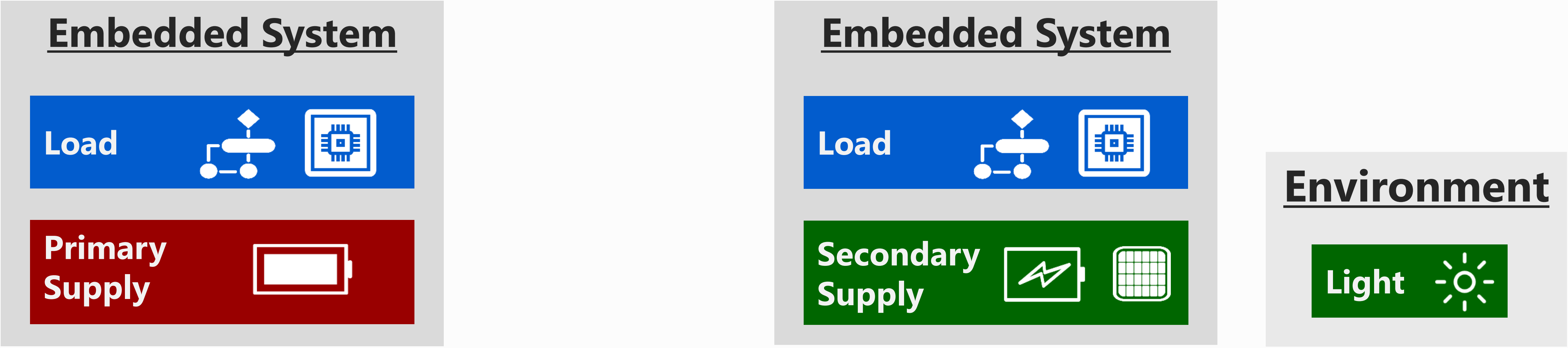


~1 cm^2

How can we power our devices in a way that: is cheap, maintenance-free

guarantees a minimum service

Two basic energy supplies:



Pros: **Continuous** energy availability

Low leakage current

Cons: **Not scalable (expensive)**

Pros: **Scalable** energy supply

Lower energy storage needs

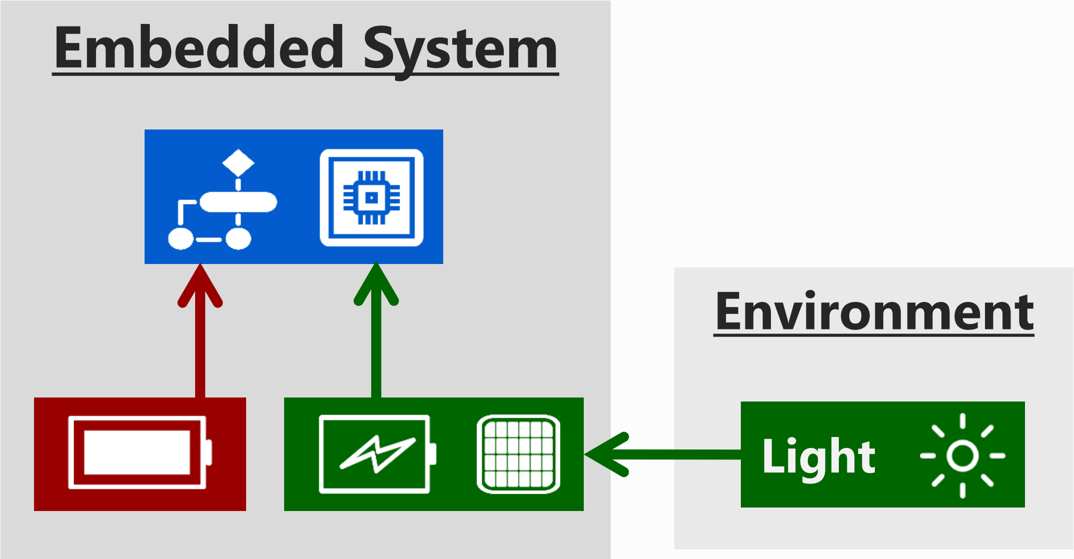
Cons: **Variable** energy availability

Limited recharge cycles

Third option: hybrid

Basic algorithm:

if then
use
else
use



Primary supply:

Expensive but guaranteed

reduce load onsumption

Secondary supply:

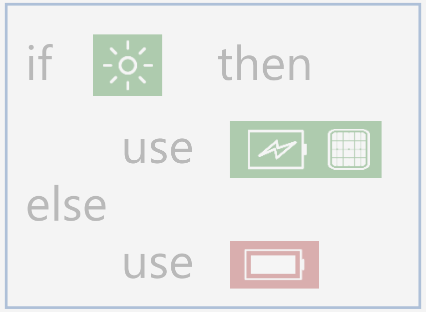
“Free” but limited and variable

minimize costs → no additional batteries

maximize efficiency

Third option: hybrid

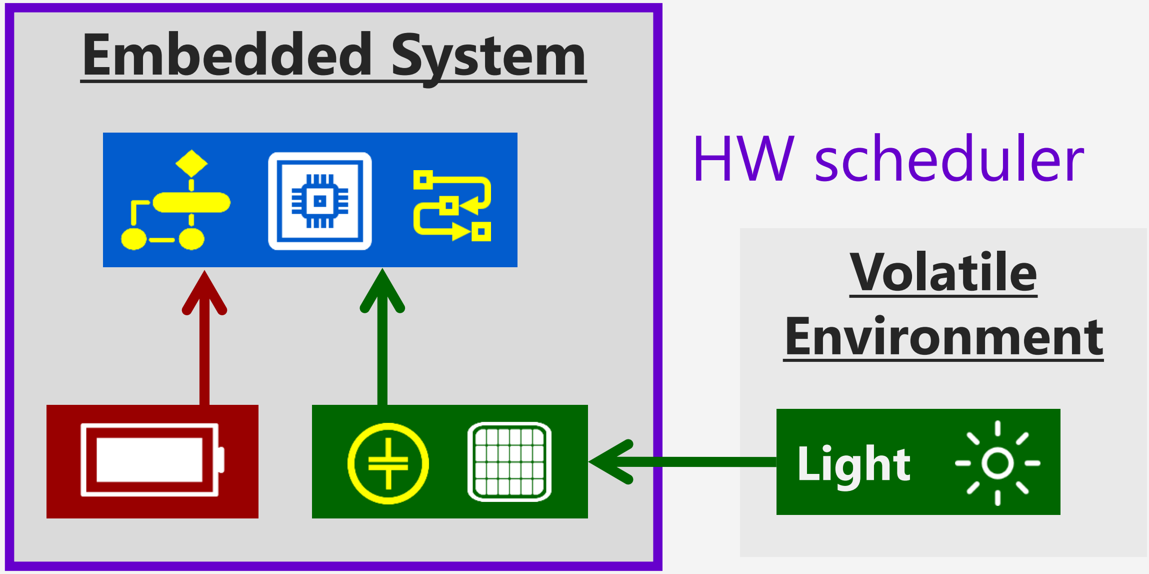
Basic algorithm:



Contributions

Task Model

HW scheduler



System architecture

Primary supply:

Expensive but guaranteed

reduce load consumption

Secondary supply:

"Free" but limited and variable

minimize costs → no additional batteries

maximize efficiency

About the authors

Patroklos Anagnostou [1]

Andres Gomez [1]

Pascal Alexander Hager [1]

Hamed Fatemi [2]

Jose Pineda de Gyvez [2]

Lothar Thiele [1]

Luca Benini [1,3]

1 ETH zürich

2 NXP

3 ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

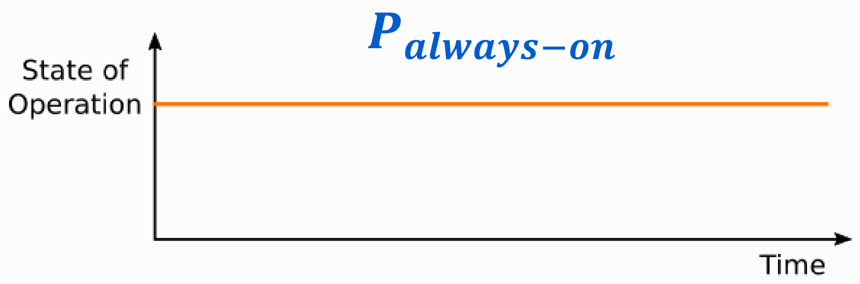
System Model & Design

Application Model

1) Always-on tasks

Continuous and low-power

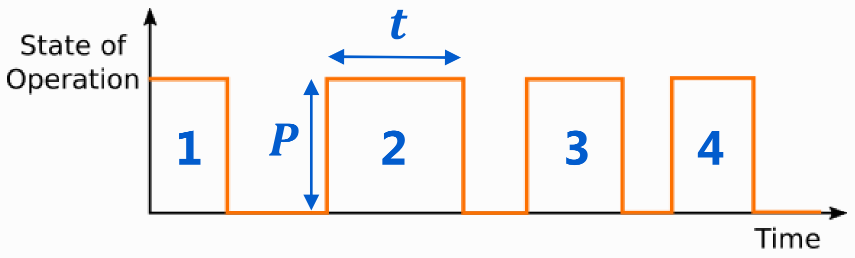
e.g. SRAM retention, timekeeping, ULP receiver



2) Energy-driven tasks

Short duration but high-power

e.g. sensing, processing, transmitting



N energy-driven tasks, each with its own P_i, t_i

The Torpor Approach

Balance the load between two supplies:

Primary ($$$ - reliable)

always-on tasks

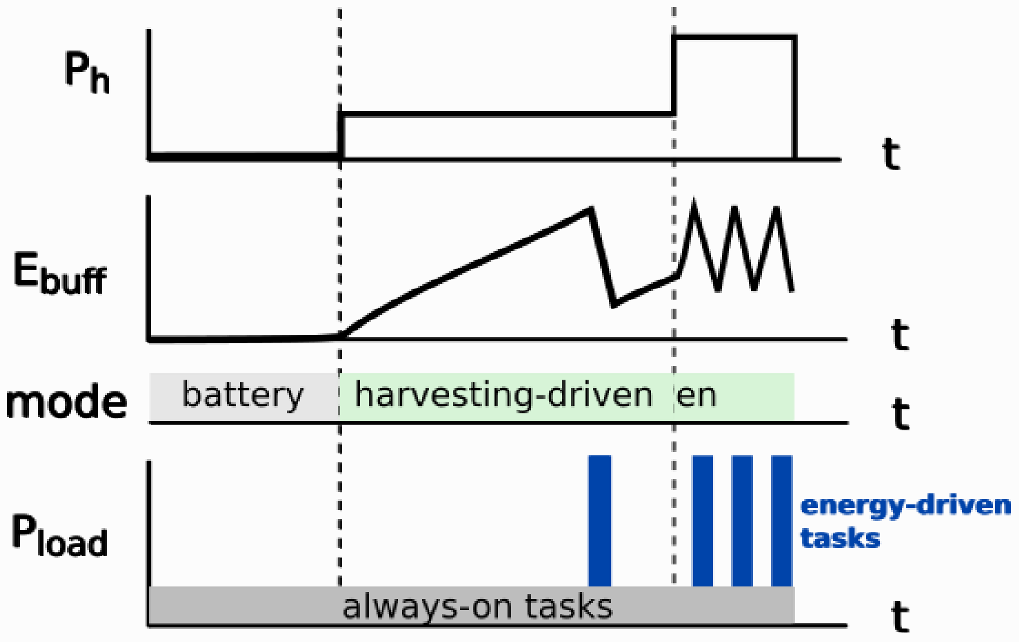
} reduced service (torpor)

Secondary ($ - unreliable)

always-on tasks

energy-driven tasks

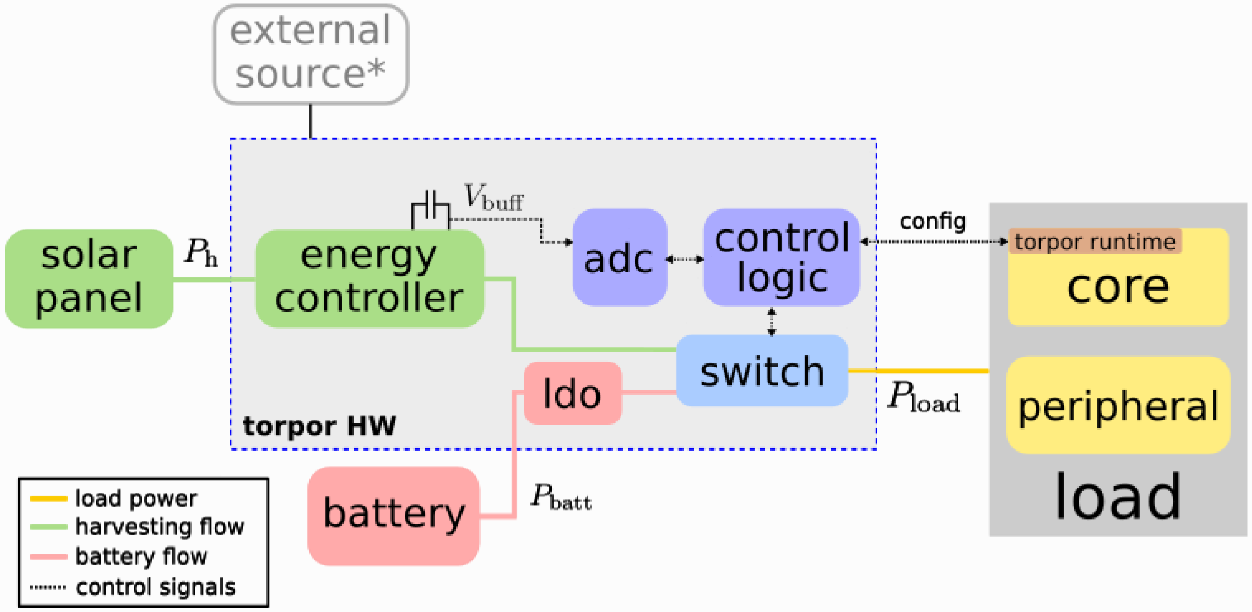
} full service (energy proportional)



Opportunistic use of harvested energy: low-cost, efficient and scalable

requires energy (or power) aware hardware

System Design



Still unsolved:

determine **task energies**

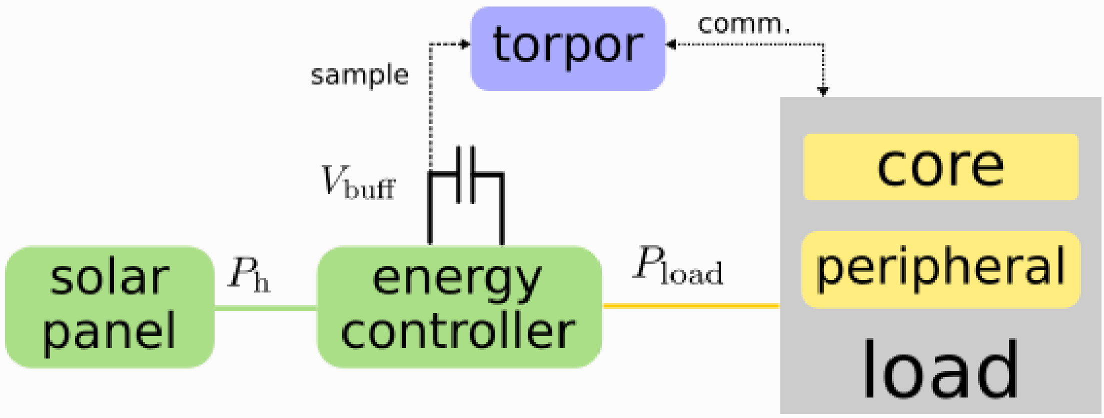
→ automated characterization

choose **which task(s)** to execute

→ dynamic scheduling

Energy Controller: [Gomez2016] *Dynamic Energy Burst Scaling for Transiently Powered Systems.*

Harvesting subsystem dynamics



Energy controller has voltage-dependent efficiency: **lower V_{buff}** is better

If $P_h(t)$ is constant: All comparable schedulers perform equally well

If $P_h(t)$ is variable: Schedulers can improve on time-dependent parameters

Scheduling Algorithms

Two energy-driven tasks:

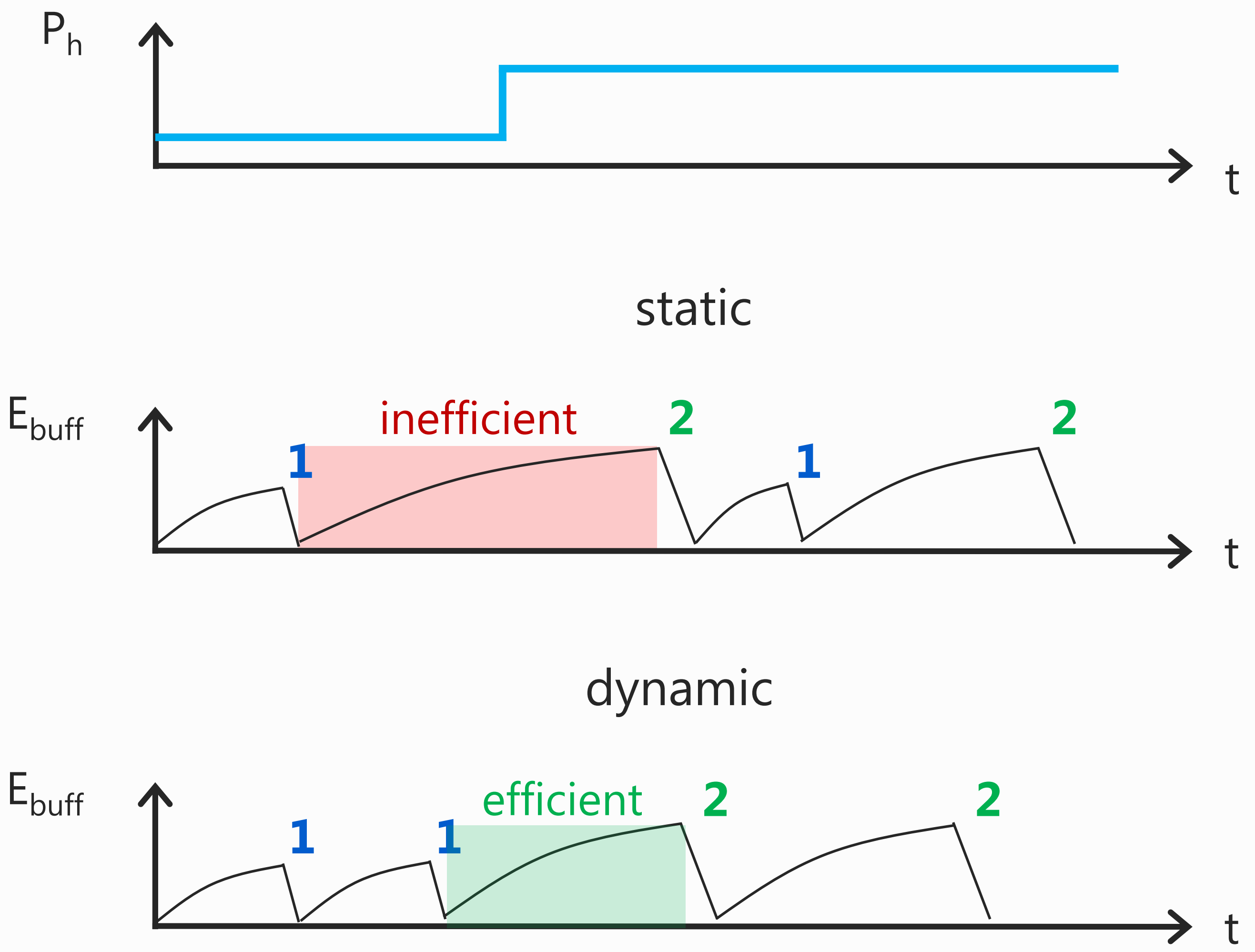
- **1** – **low** energy
- **2** – **high** energy

Static scheduling:

- Determined **offline**
- Requires only voltage threshold

Dynamic scheduling:

- Determined **online**
- Requires monitoring voltage history



Generalized scheduling

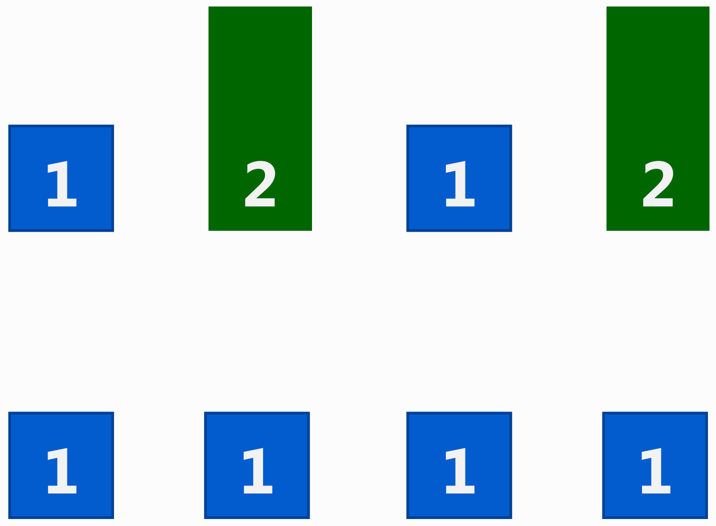
Torpor implements priority-based scheduling

Simplified algorithm:

maximize throughput

if $P_h < P_{critical}$

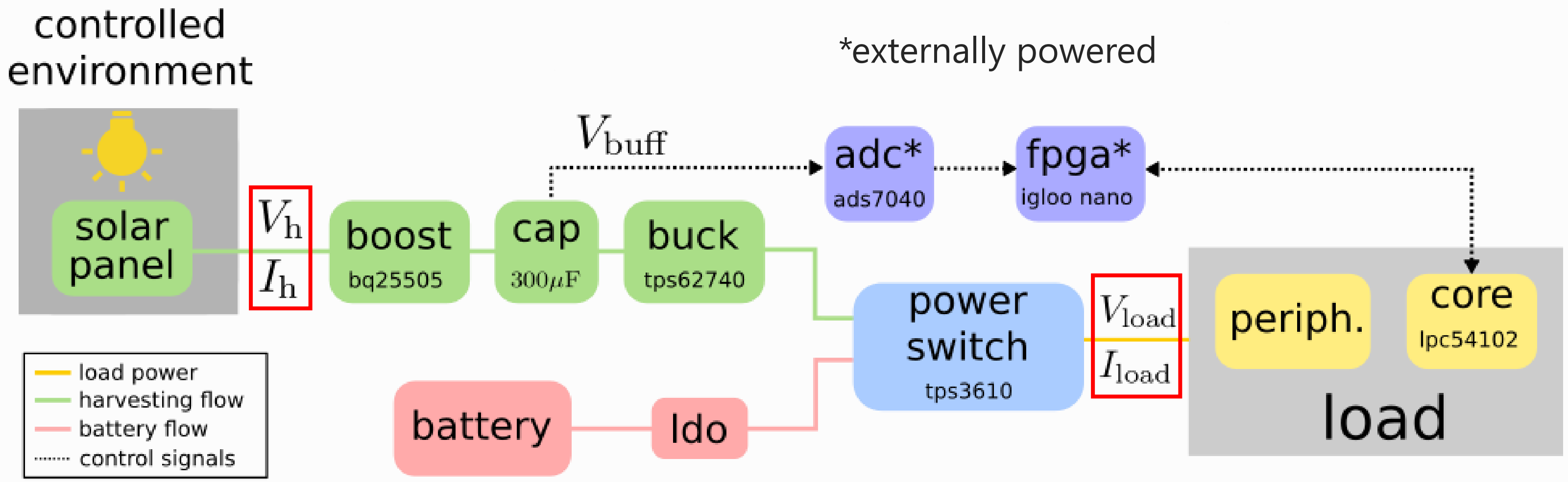
minimize spent energy



Torpor can adjust priority according to buffer states

Experimental Evaluation

Experimental Setup



4 different input power traces

2 constant, 2 variable

3 different schedulers

2 static, 1 dynamic

3 different synthetic task-sets

Energy driven: 7-90 mW (S+P+T applications)

Always on: 600 µW (LPM0)

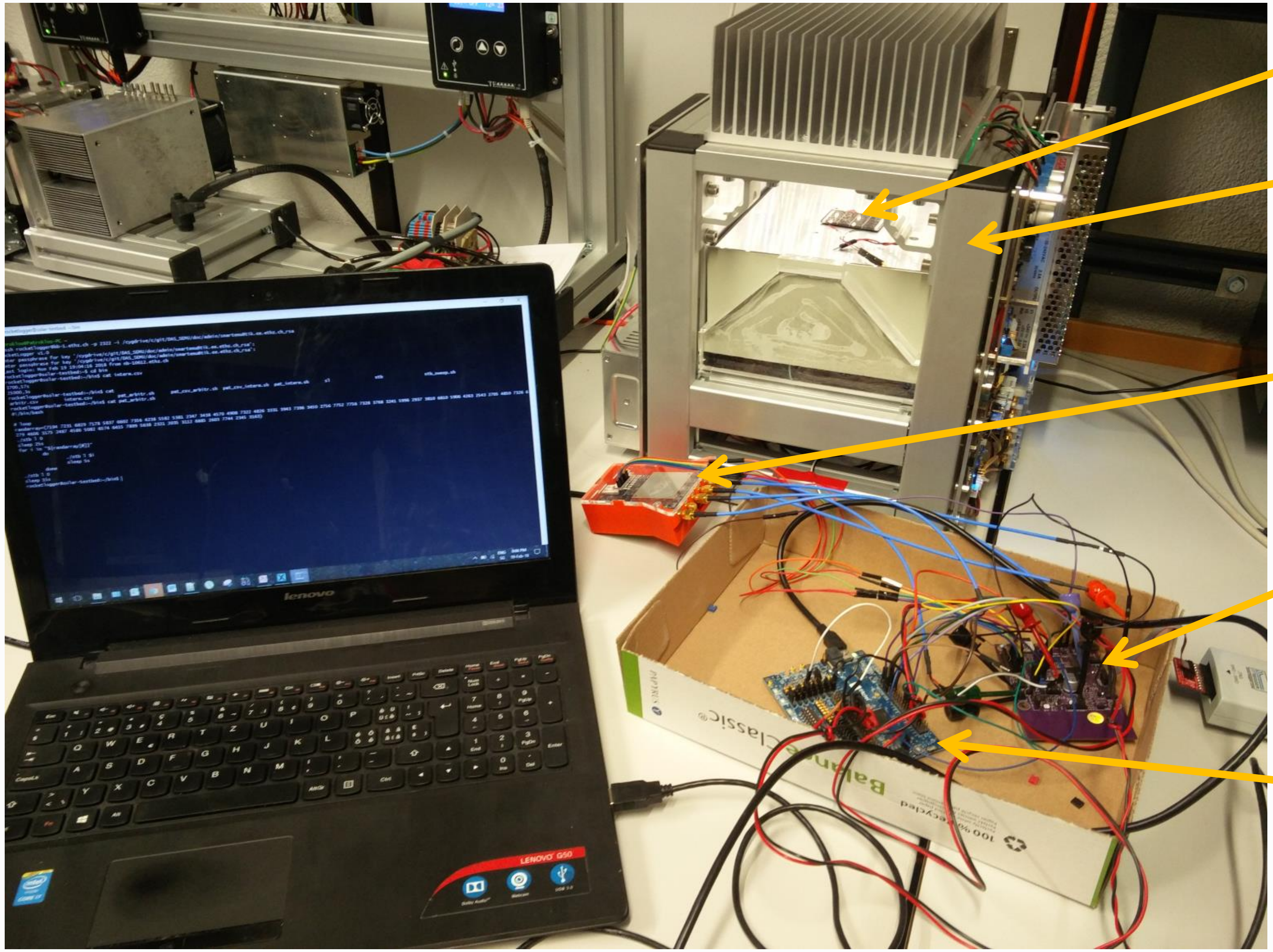
Performance evaluation :

Executions/minute

Average P_{load}

Energy efficiency $\left(\frac{E_{load}}{E_h}\right)$

Experimental Setup



Solar Panel

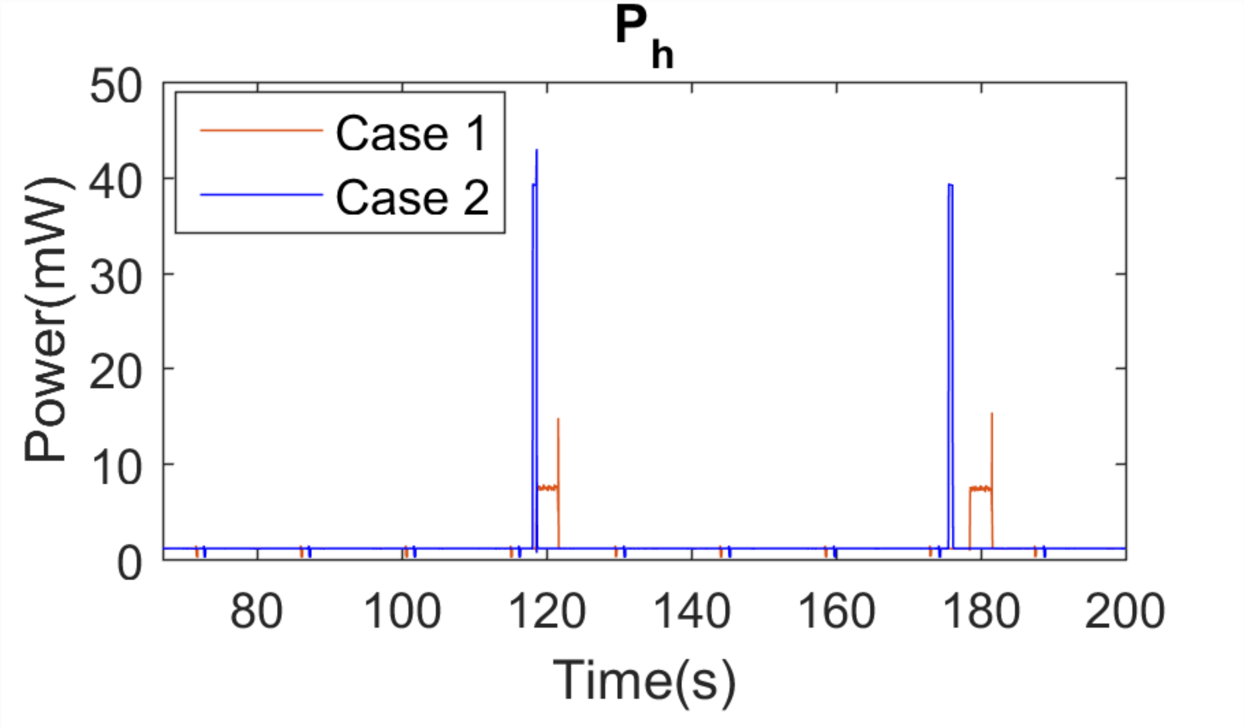
Solar Testbed

RocketLogger

HTVNv2 board

FPGA Board

Static vs Dynamic Schedulers



Execution Rate [execs/min]

	Static Split	Dynamic	Improvement
Case 1	3.0	3.7	x1.3
Case 2	2.8	6.2	x2.2

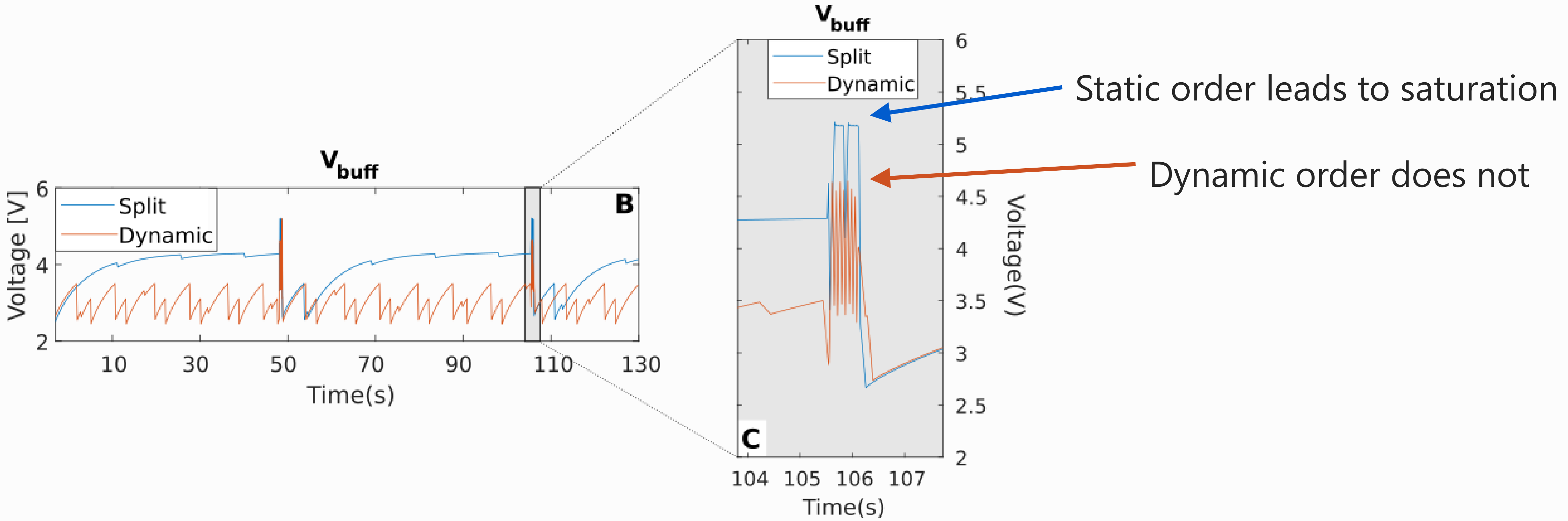
Energy Efficiency [%]

	Static split	Dynamic	Improvement
Case 1	60.6	66.4	x1.1
Case 2	61.2	69.7	x1.1

Dynamic scheduling adjusts better to highly variable environments

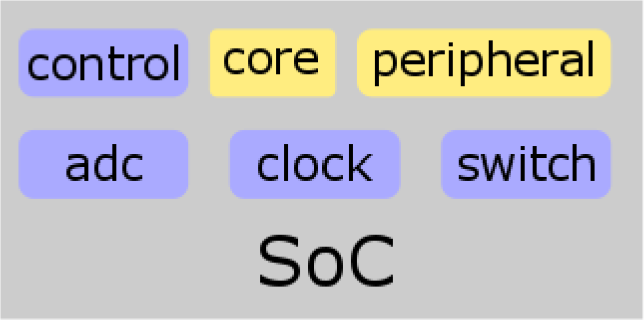
Avoiding Saturation

Even though $\overline{P}_h < \max(P_{load})$, for a short time $P_h > \overline{P}_{load}$



Highly variable task sets need power-aware scheduling

Torpor Overheads

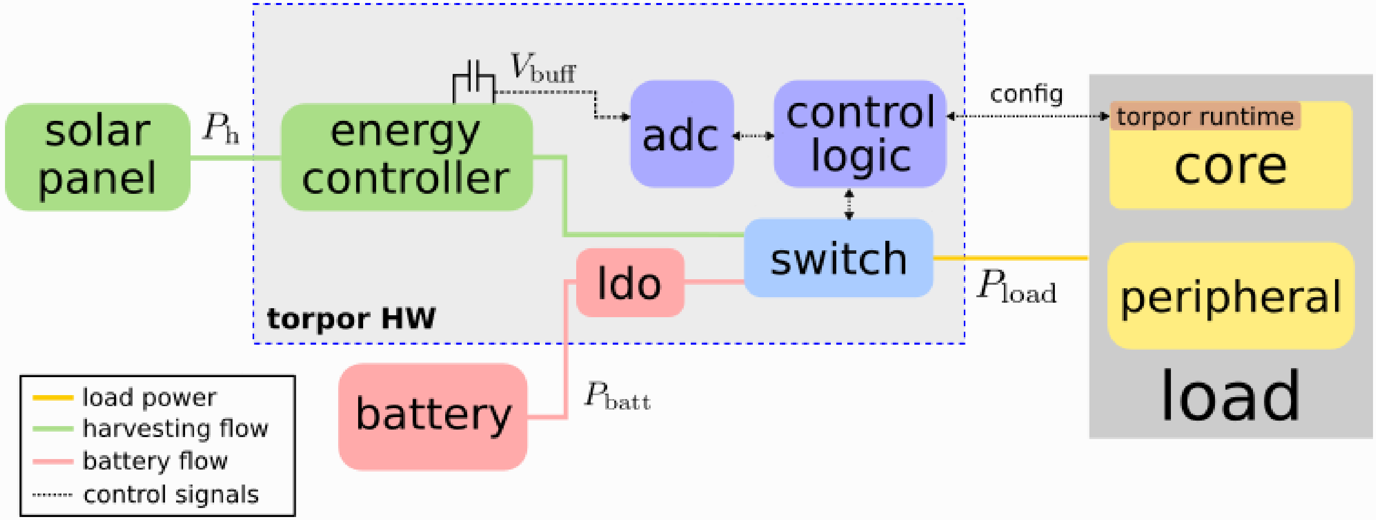


- Torpor logic was synthesized
  - GF 22nm technology, 0.65V, TT case, 25°C
  - Operational frequency: 32 KHz – 2.4 MHz
- Estimated area (control only): 1700 μm^2
- Clock, ADC, Switch deduced from components

Torpor Module	Power
Control	1.57 – 1.96 µW
Clock	42 nW
ADC	0.2 – 1 µW
Power switch	800 nW
Total	**< 4 µW**

comparable to deep sleep modes

Summary



Torpor: reduced service mode during energy unavailability

Static schedulers work well when harvested power is constant

Dynamic schedulers keep working well under volatile harvesting conditions

HW implementation consumes < 4 μW

can double execution rate (compared to static schedulers)

BACKUP

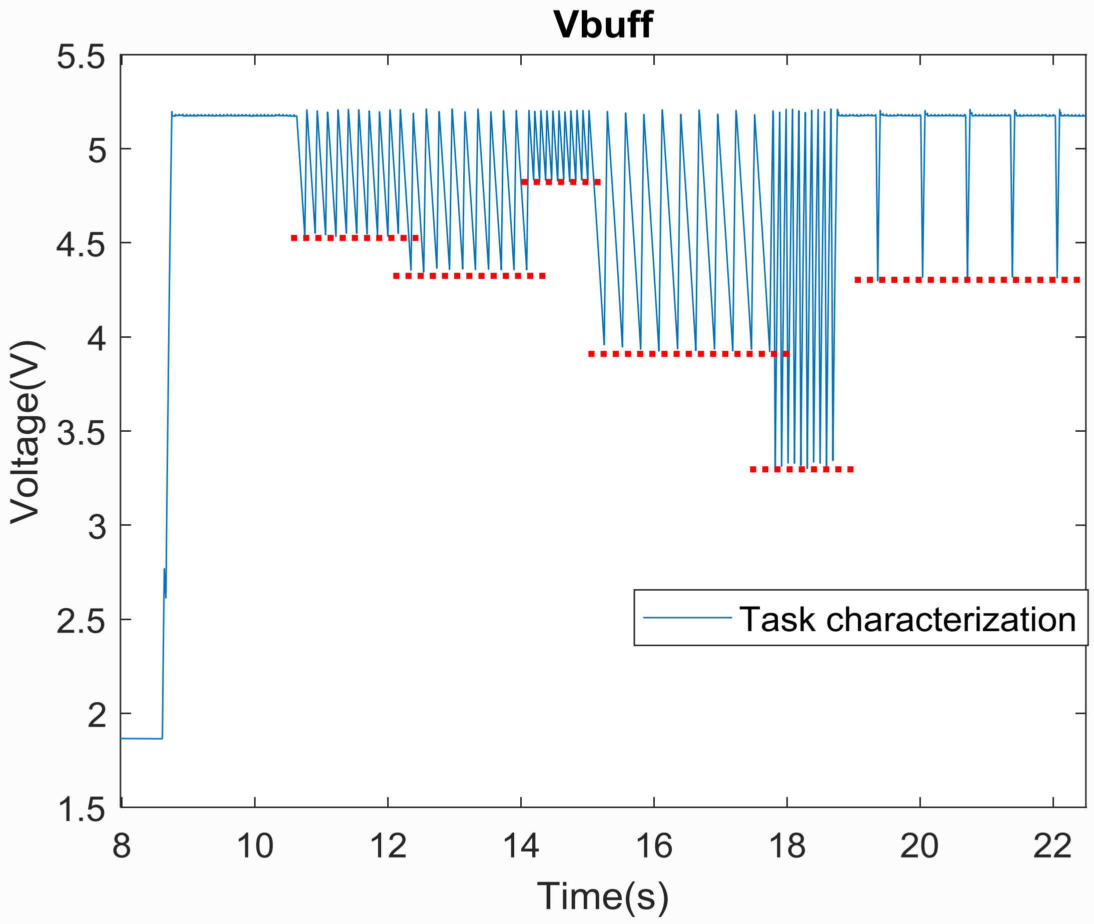
Task Characterization

To find voltage thresholds for each task

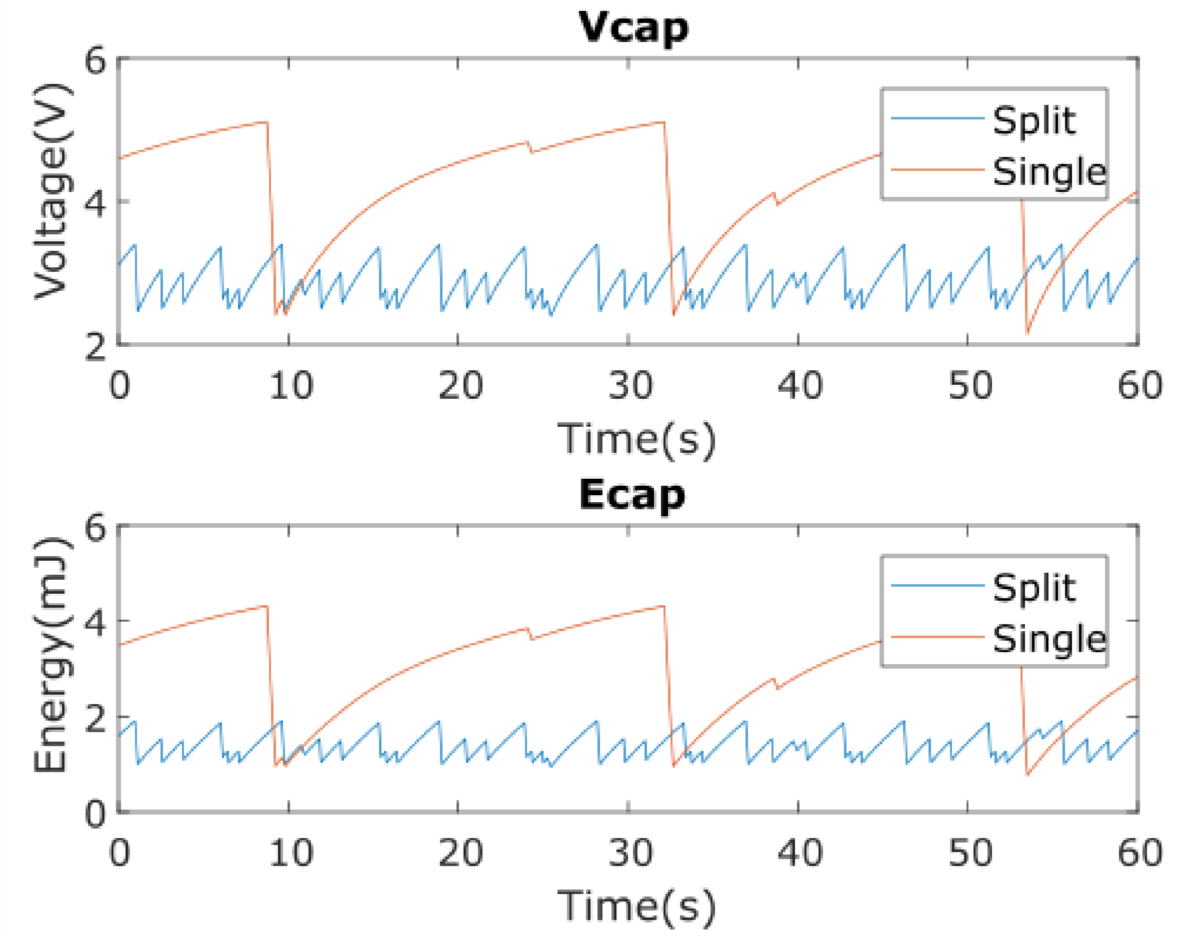
- theoretical formulas
- manual trial & error attempts

Torpor aids empirical characterization:

1. Charge the capacitor to maximum
2. Cut-off input power and execute task
3. Measure voltage through ADC
4. Repeat and average measurements
5. Calculate the thresholds required

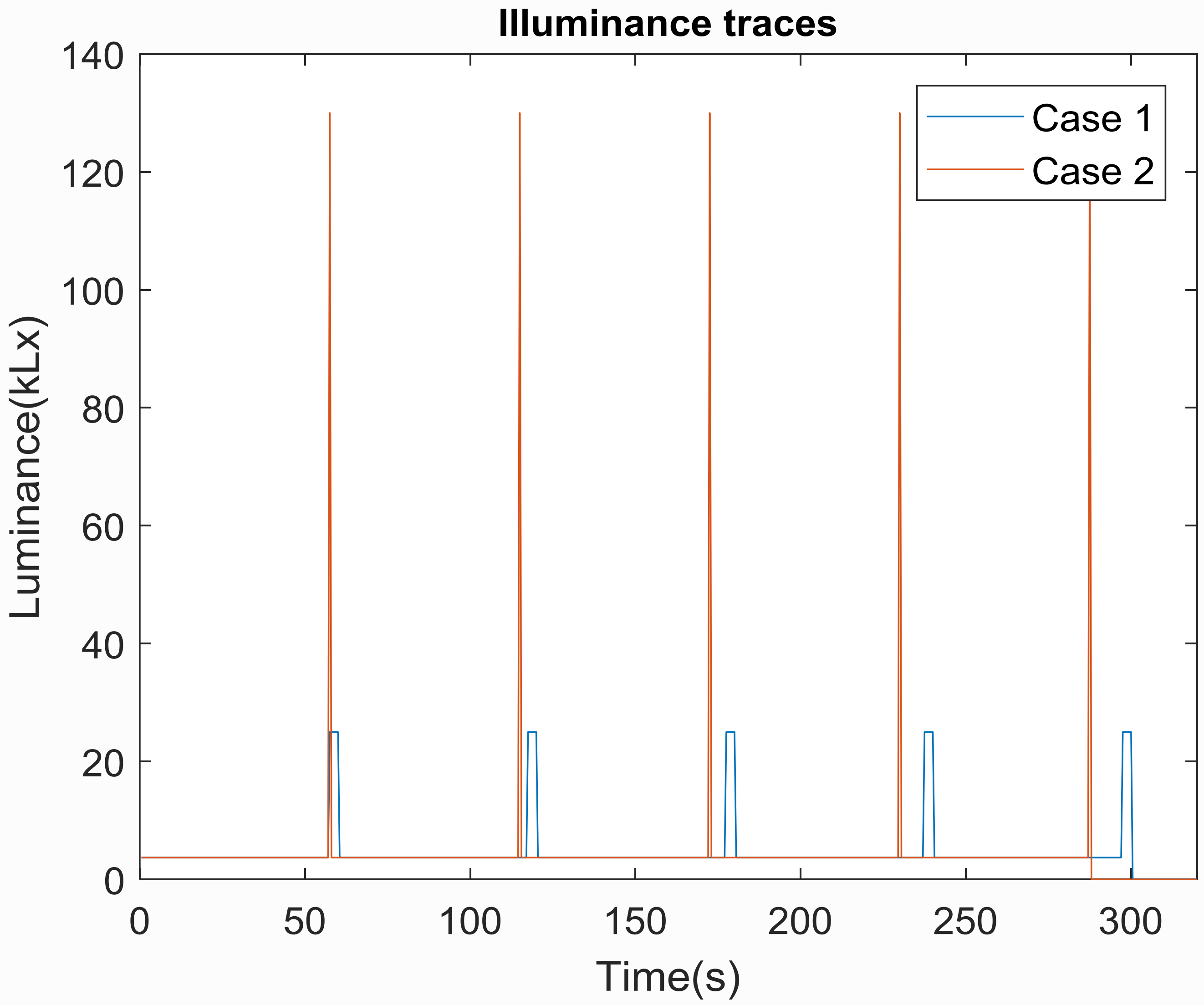


Static schedulers



	$P_h = 1.3$ [mW]			$P_h = 3.2$ [mW]		
	scheduler		**ratio**	**scheduler**		**ratio**
	single	split		single	split	
exec/min	2.6	6.9	×2.6	31.2	39.1	×1.3
E_{eff} [%]	55.2	67.0	×1.2	58.0	66.7	×1.2
$\bar{P}_{\text{load}}$ [mW]	0.71	0.87	×1.2	1.84	2.16	×1.2

Variable input power



Scheduling in HW *and* SW

Always on, must be efficient

HW

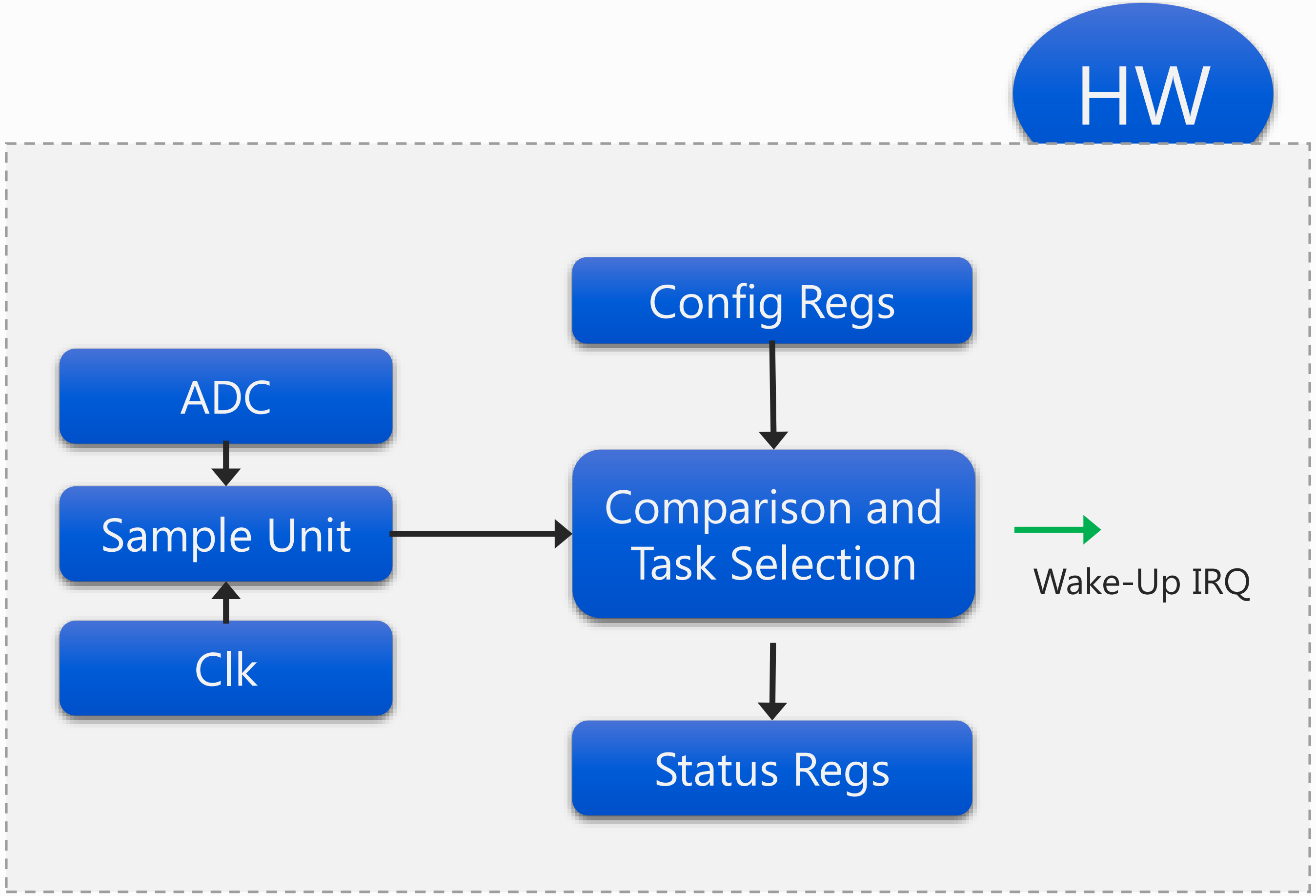
- Voltage monitoring (E_{buff}, P_h)
- Wake-up logic
- Tasks selection according to priorities, E_{buff}, P_h

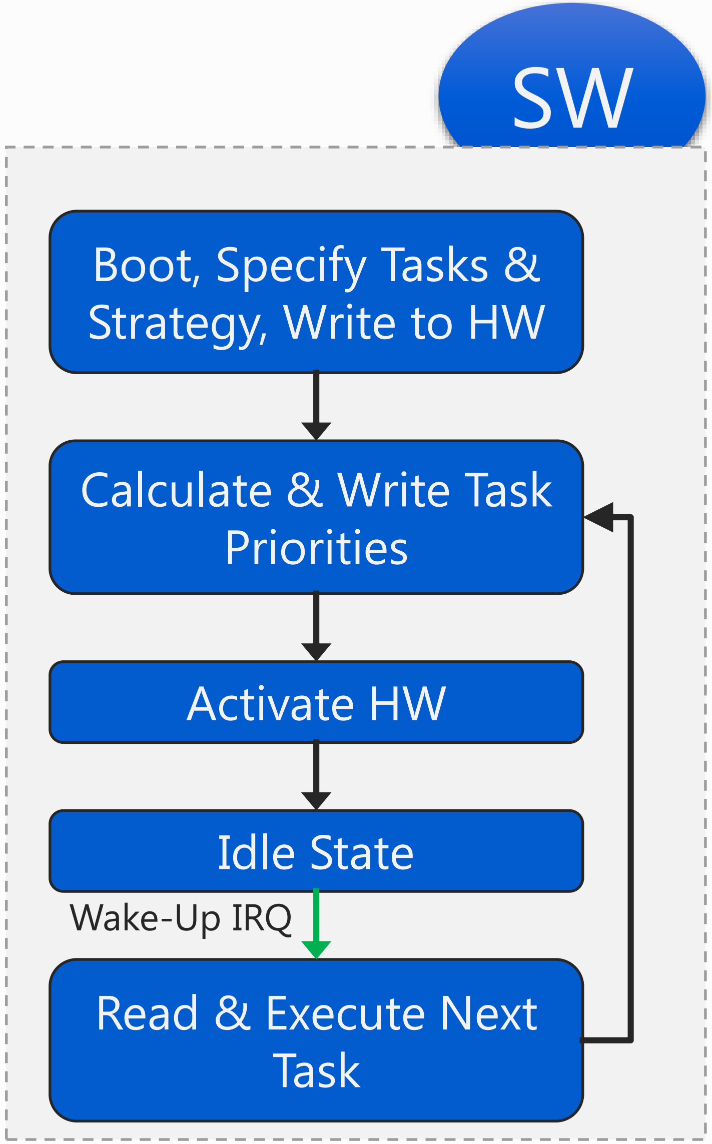
Configurable, extensible, user-friendly

SW

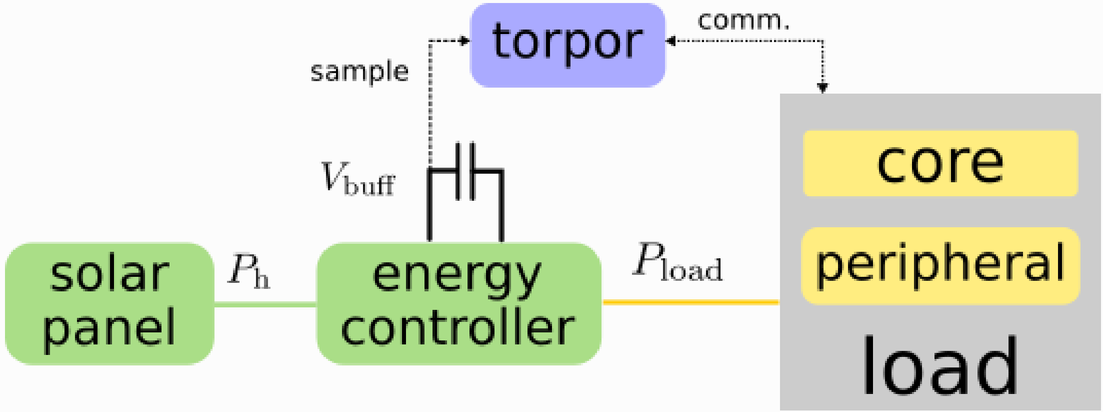
- Task declaration
- Configuration (Policy, Thresholds, etc)
- Task priorities update according to policy and state of software

Torpor Logic





Harvesting subsystem dynamics



environment, hardware, scheduler

$$\frac{d}{dt}E_{buff}(t) = \underbrace{P_h(t)}_{\text{environment}} - \underbrace{P_{load}(S_i)}_{\text{scheduler}} - \underbrace{\overline{P}_{torpor}}_{\text{hardware}}$$

Energy controller has voltage-dependent efficiency: **lower V_{buff}** is better

If $P_h(t)$ is constant: All comparable schedulers perform equally well

If $P_h(t)$ is variable: Schedulers can improve on time-dependent parameters