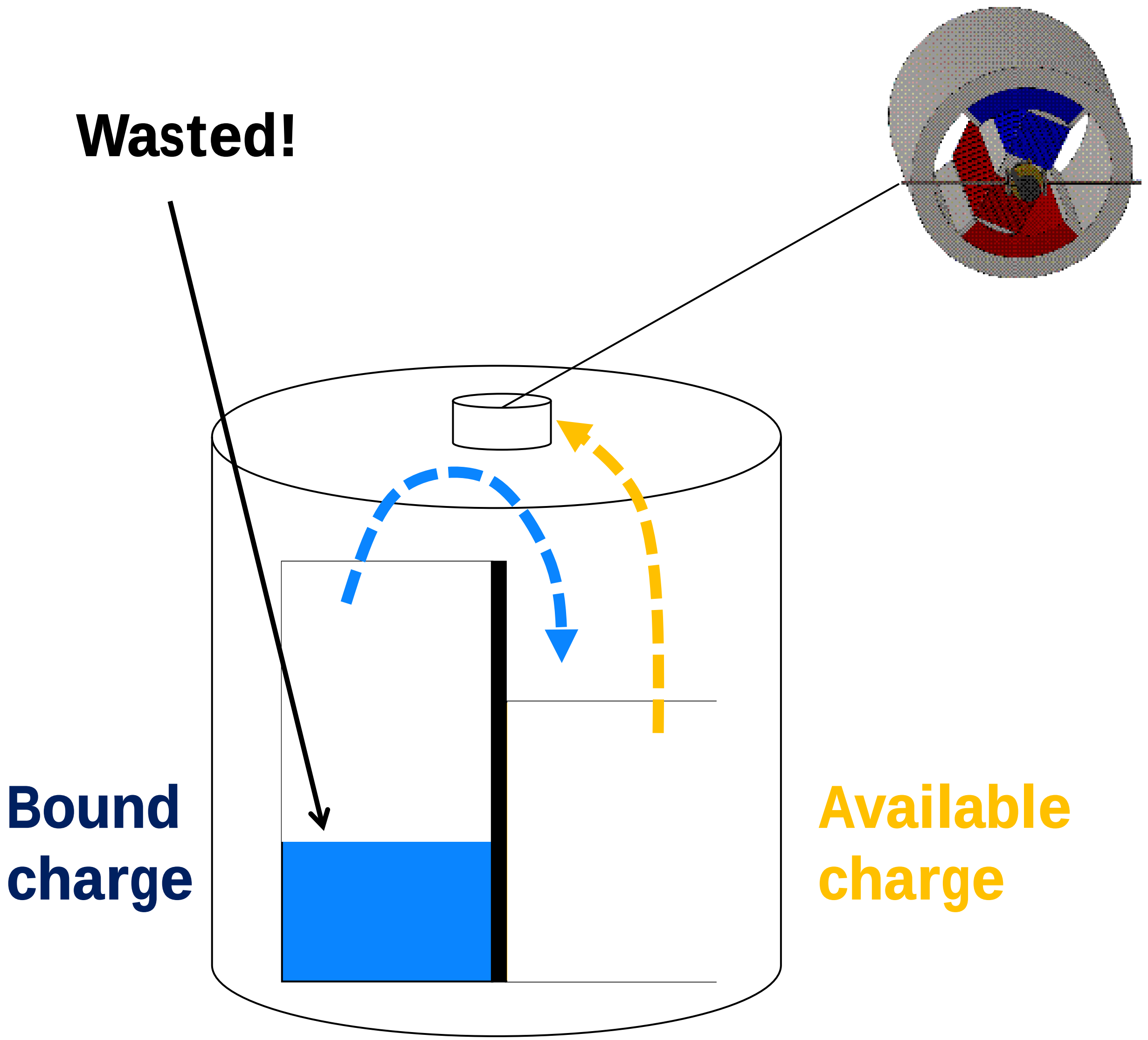
Automatic Construction of Efficient Multiple Battery Usage Policies

Maria Fox   Derek Long
University of Strathclyde, UK

Daniele Magazzeni
University of Chieti-Pescara, Italy

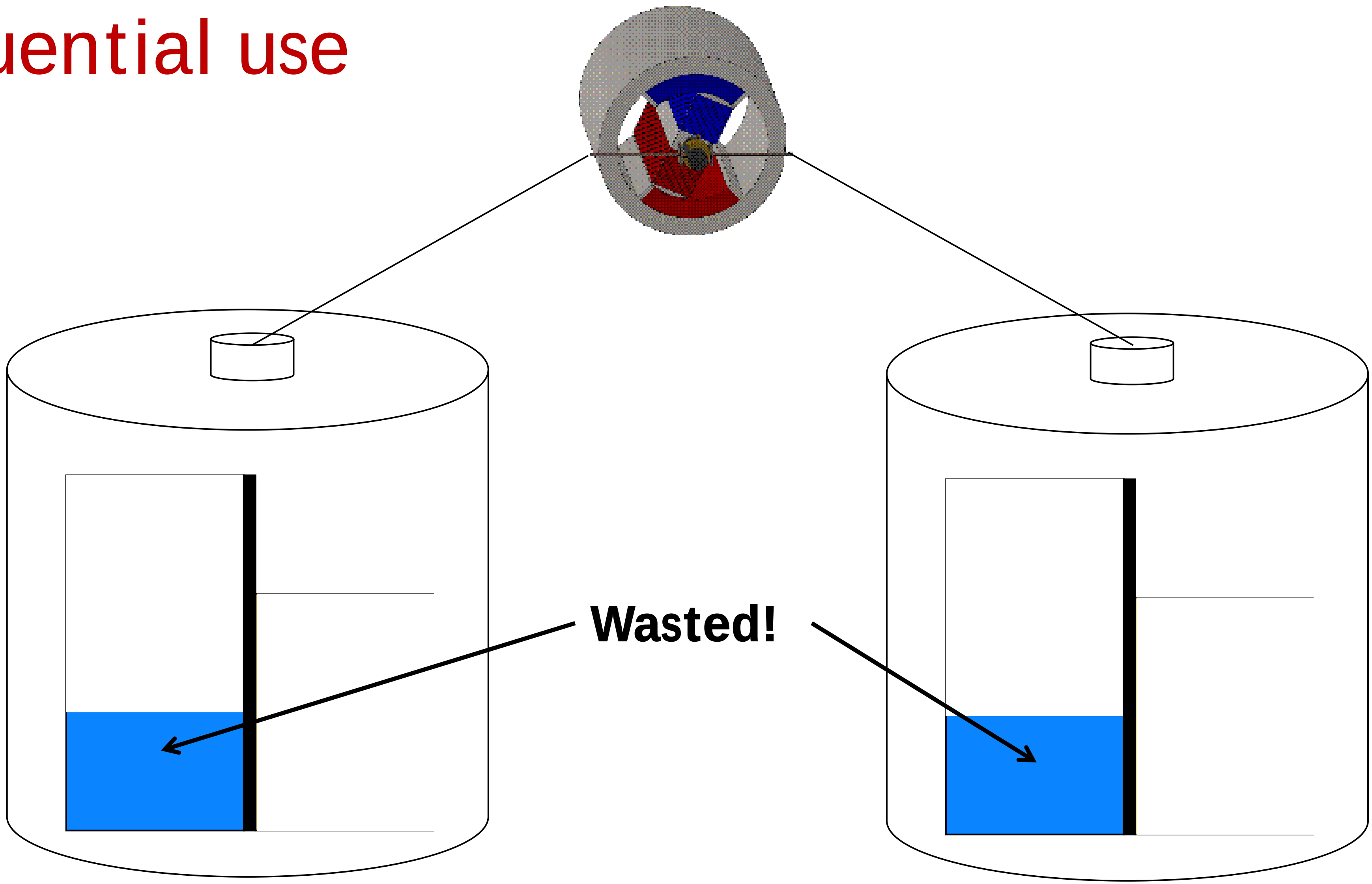
Motivations

A brief introduction to batteries…



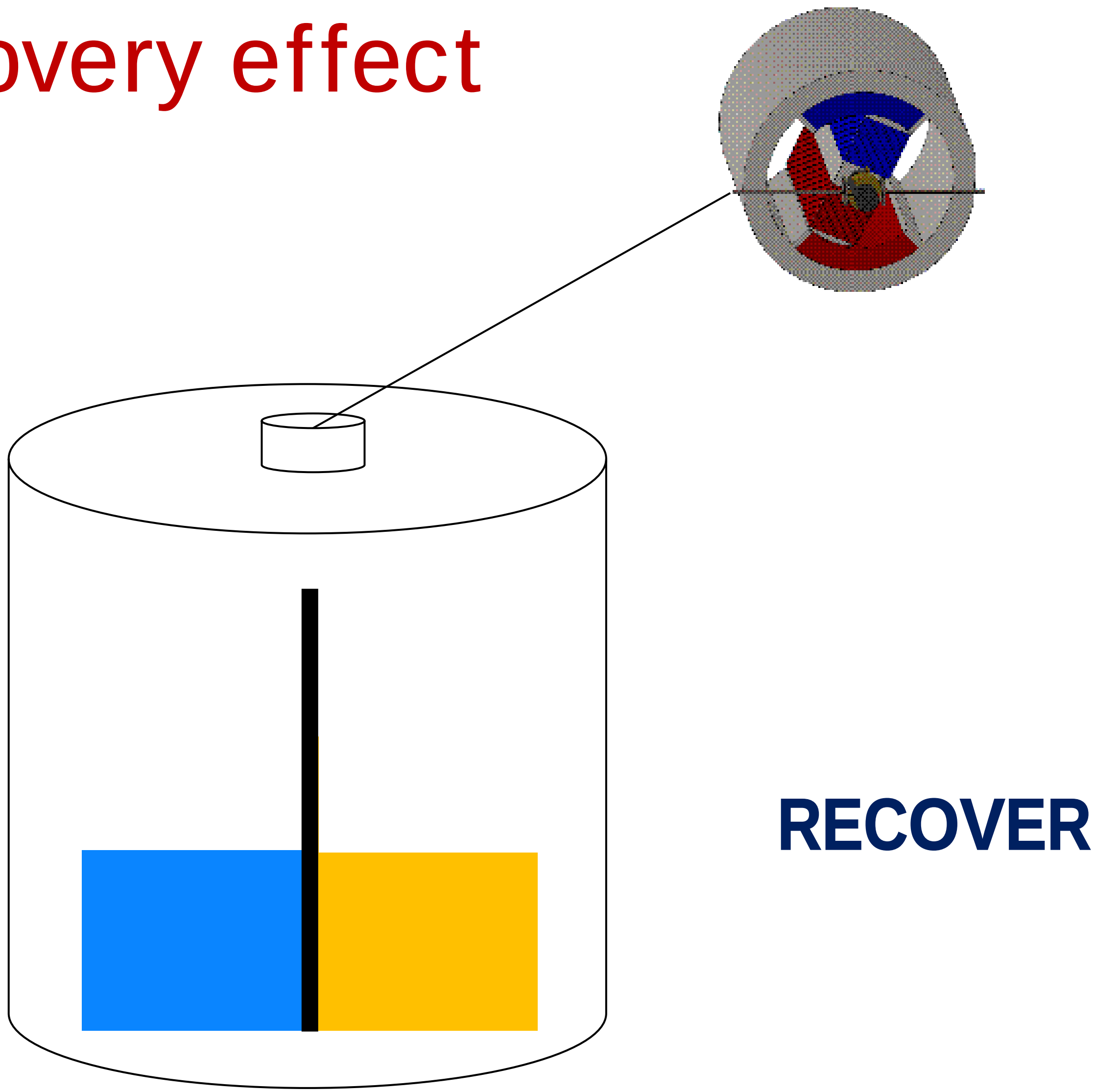
A brief introduction to batteries…

Sequential use



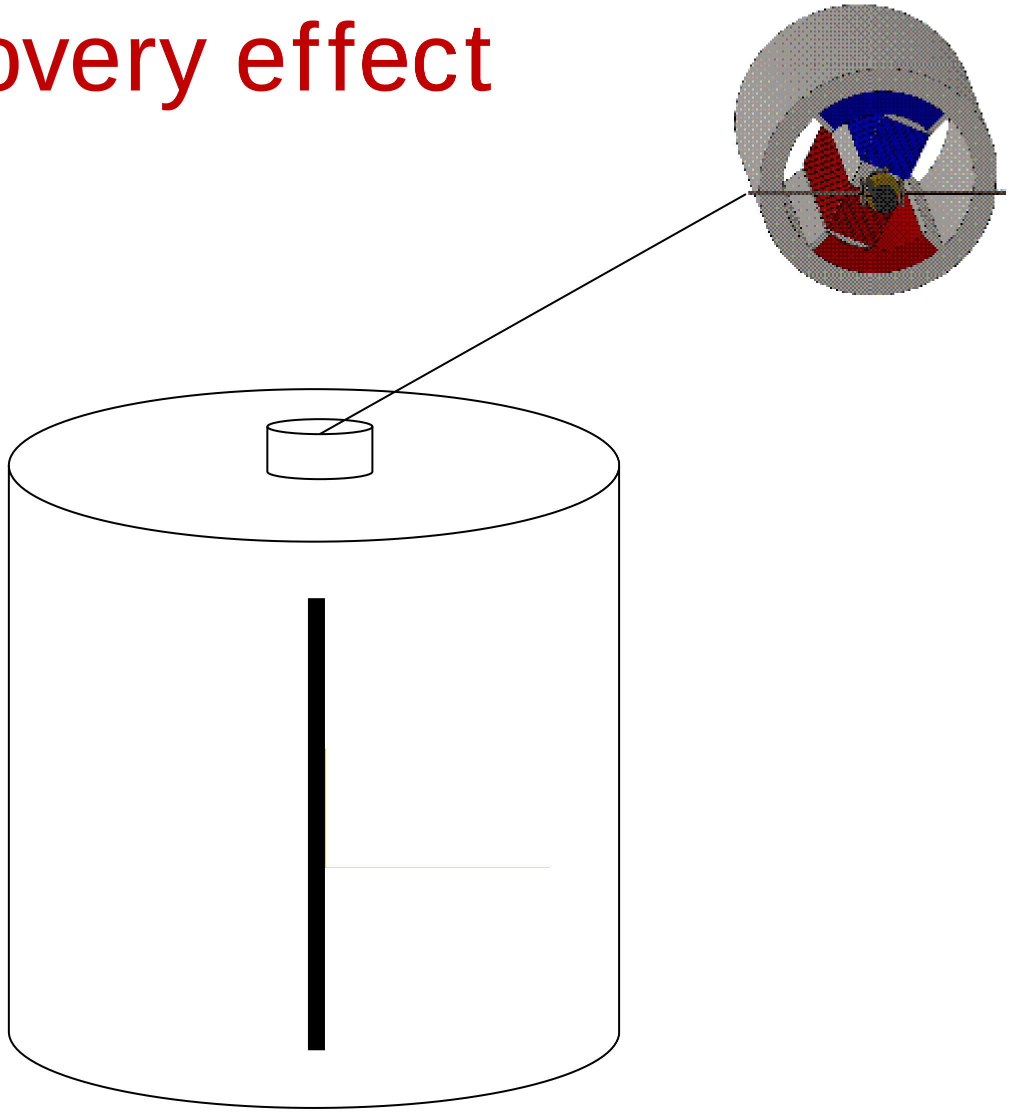
A brief introduction to batteries…

Recovery effect



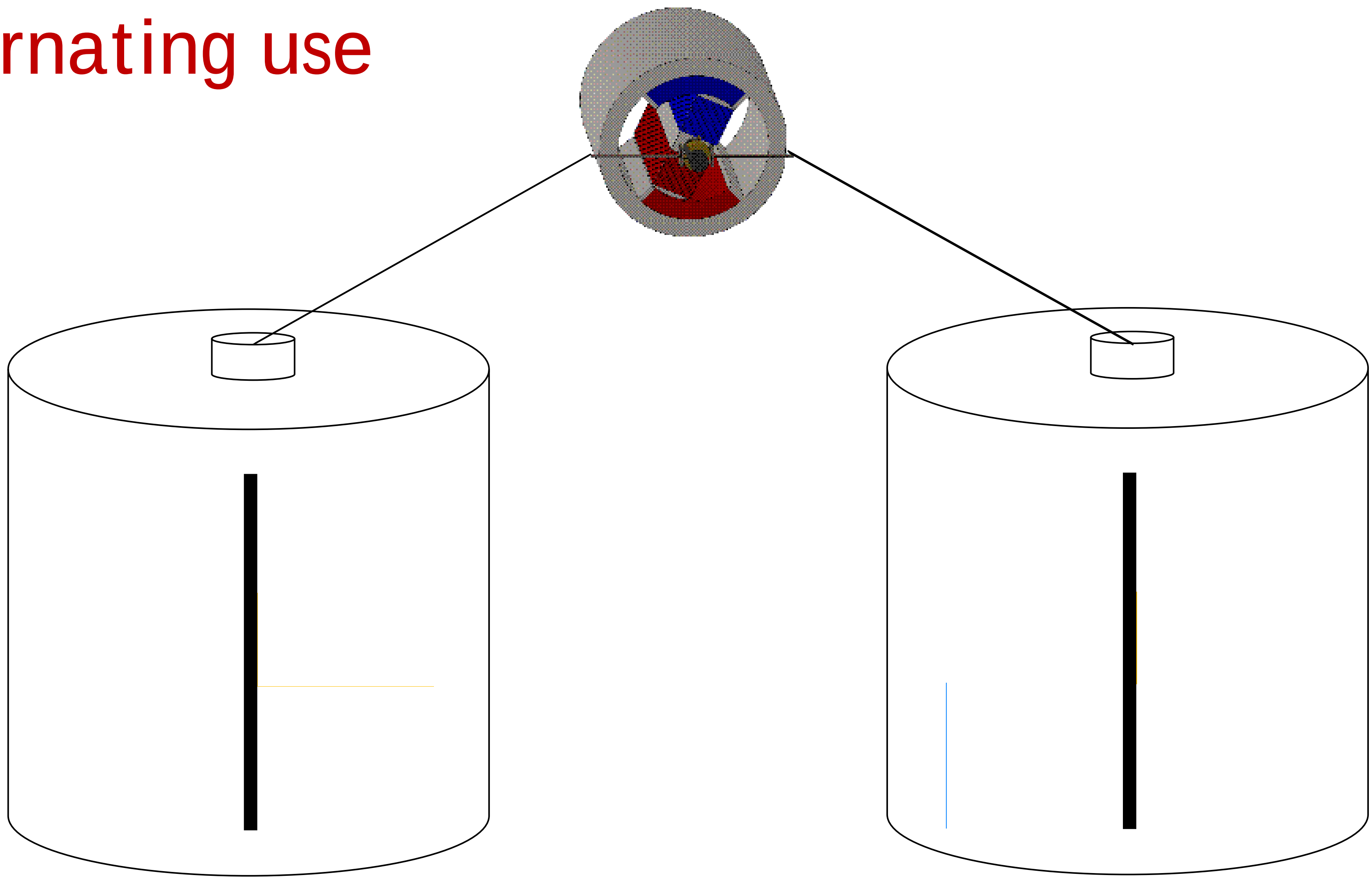
A brief introduction to batteries…

Recovery effect



A brief introduction to batteries…

Alternating use



Multiple Battery Usage Problem

- Problem explored from electrical engineering perspectives
- Efficient use of batteries lies in the design of effective policies managing the switching of load between them
- A new battery is selected whenever the voltage of the battery currently servicing the load drops below a threshold
  - V_{MAX}: select the battery pack with highest state of charge
  - V_{MIN}: select the battery pack with lowest state of charge
  - T_{MAX}: select the battery pack that has been unused for the longest time
  - T_{MIN}: select the battery pack that has been unused for the shortest time

The best one, as shown by *Benini et al.*

Multiple Battery Usage Problem

- Jongerden et al (2009) uses a model checking strategy (UPPAAL) to schedule battery use given a known load profile
- The approach is effective but does not scale well

- Standard policies are typically simple, based on rapid switching between available batteries
- An optimal policy can be obtained theoretically by switching between batteries at extremely high frequency
- Such a policy is not achievable in practice
- The resulting lifetime represents a *theoretical upper bound*

Our Contribution

- We consider probability distributions for load size, load duration and load frequency
- We use a well-established continuous battery model (KiBaM)
- A non-linear mixed discrete-continuous optimisation problem
- We propose an approach based on:
  - Heuristic search
  - Machine learning (classification)

- The best deployed solutions deliver less than 80% efficiency
- The best published solutions deliver less than 95% efficiency
- We produce robust solutions that deliver better than 99% efficiency

The Kinetic Battery Model

- Proposed by Manwell and McGowan (1994)
- Used by Jongerden et al. (2009)

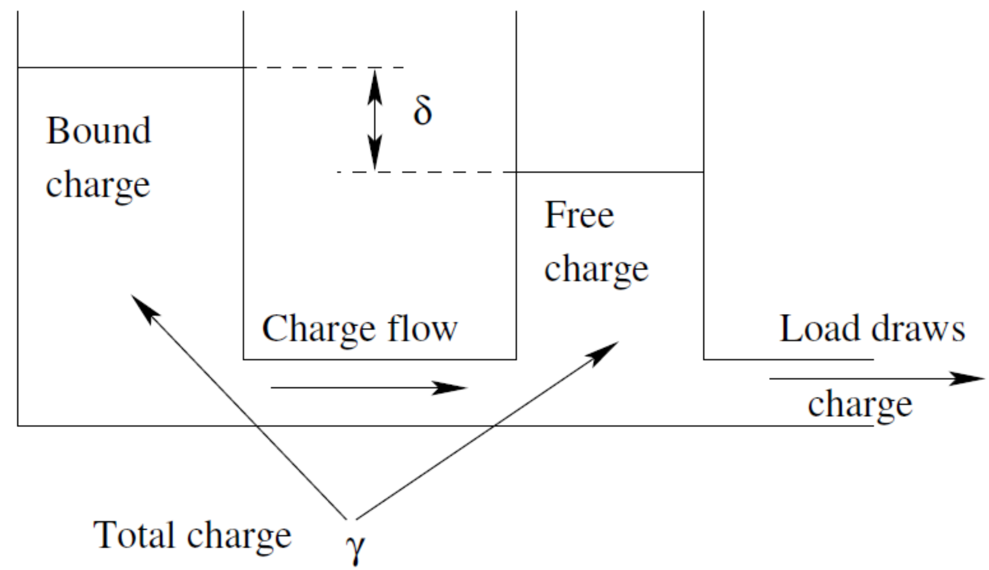
$$\frac{d\delta}{dt} = \frac{i(t)}{c} - k'\delta$$

$$\frac{d\gamma}{dt} = -i(t)$$

load

Fixed conductance

battery capacity



$$\delta(t) = \frac{I}{c} \cdot \frac{1 - e^{-k't}}{k'}$$

$$\gamma(t) = C - It$$

BUP as MDP

- BUP can be cast as *hybrid temporal Markov Decision Process*
- Battery switching actions are deterministic, but the events that cause load to change are not
- Time between events is governed by a stochastic process, but timing of switching actions is deterministic

$$(sb_1, sa_1, sb_2, sa_2, \ldots, sb_n, sa_n, B, t, L)$$

bound charge (sb_1)

available charge (sa_1)

battery currently in use (B)

time of the state (t)

current load (L)

BUP as MDP

- BUP can be cast as *hybrid temporal Markov Decision Process*
- Battery switching actions are deterministic, but the events that cause load to change are not
- Time between events is governed by a stochastic process, but timing of switching actions is deterministic

$$(sb_1, sa_1, sb_2, sa_2, \ldots, sb_n, sa_n, B, t, L)$$

Use B’

$$(sb_1, sa_1, sb_2, sa_2, \ldots, sb_n, sa_n, B', t, L)$$

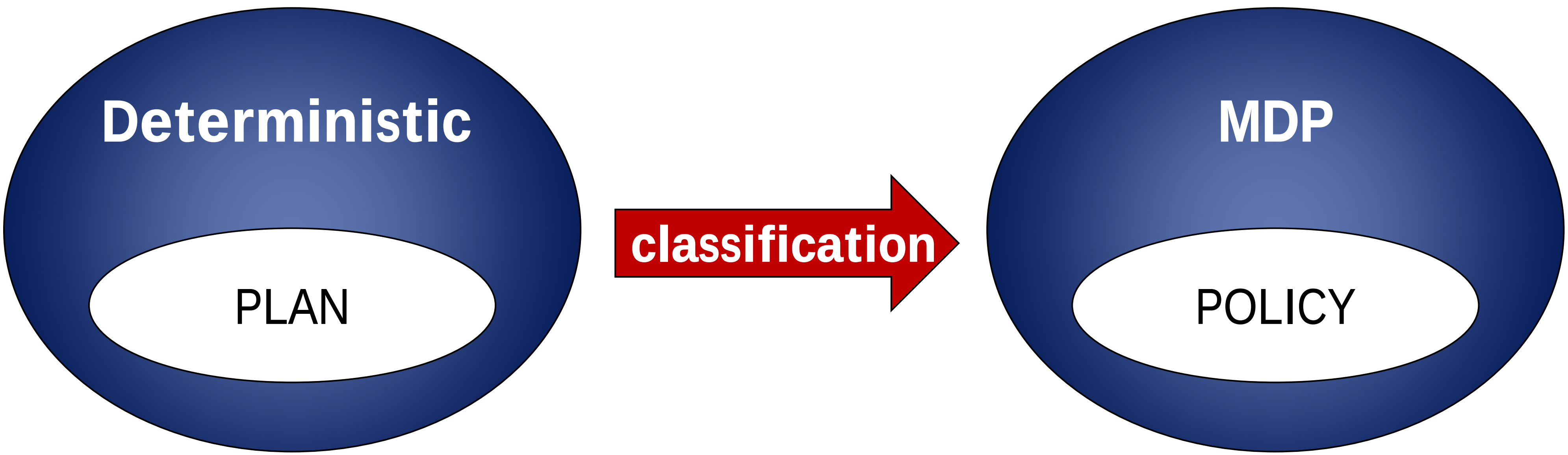
Wait(T)

$$(sb_1, sa_1, sb_2, sa_2, \ldots, sb_n, sa_n, B, t+T, L)$$

BUP as MDP (related work)

- Hybrid AO* search (Meuleau et al. 2009)
  - A dynamic programming approach to guide heuristic search for problems involving continuous resources used by stochastic actions
  - It does not handle time-dependent resource consumption
  - Empirical data for solution of problems with up to 25,000 states
- Mausam and Weld (2008)
  - Planner for concurrent MDPs (with temporal uncertainty)
  - It does not manage continuous time-dependent resources
  - Empirical data for solution of problems with up to 4,000,000 states (>1 hour)
- Fern, Yoon, Givan (2006)
  - Machine learning applied to policy rollout samples

Our Approach



Solving Deterministic Multiple Battery Problems

PDDL+

- PDDL extension for describing mixed discrete continuous domains
- **Continuous processes**: these are active whenever their preconditions are true. They increase or decrease numeric values while the system remains in a fixed logical state over time.
- **Controllable discrete events**: these change the logical state of the system. They can be applied when their preconditions are true and their post-conditions describe their effects.
- **Uncontrollable discrete events**: these change the logical state of the system when numeric values pass critical thresholds.

A PDDL+ Battery Model

```
(:durative-action use
 :parameters (?b - battery)
 :duration (>= ?duration 0)
 :condition (and (at start (switchedOff ?b)) (over all (switchedOn ?b)))
 :effect (and (at start (not (switchedOff ?b))) (at start (switchedOn ?b))
              (at end (not (switchedOn ?b))) (at end (switchedOff ?b))))
```

The battery to be used

```
(:process consume
 :parameters (?b - battery)
 :precondition (switchedOn ?b)
 :effect (and (decrease (gamma ?b) (* #t (load)))
              (increase (delta ?b) (* #t (/ (load) (cParam ?b))))))

(:process recover
 :parameters (?b - battery)
 :precondition (>= (delta ?b) 0)
 :effect (and (decrease (delta ?b) (* #t (* (kprime ?b) (delta ?b))))))

(:event batteryDead
 :parameters (?b - battery)
 :precondition (and (switchedOn ?b) (<= (gamma ?b) (* (- 1 (cParam ?b)) (delta ?b))))
 :effect (and (not (switchedOn ?b)) (dead ?b)))
```

A PDDL+ Battery Model

Duration is decided by the planner

```
(:durative-action use
 :parameters (?b - battery)
 :duration (>= ?duration 0)
 :condition (and (at start (switchedOff ?b)) (over all (switchedOn ?b)))
 :effect (and (at start (not (switchedOff ?b))) (at start (switchedOn ?b))
                          (at end (not (switchedOn ?b))) (at end (switchedOff ?b))))
(:process consume
 :parameters (?b - battery)
 :precondition (switchedOn ?b)
 :effect (and (decrease (gamma ?b) (* #t (load)))
                          (increase (delta ?b) (* #t (/ (load) (cParam ?b))))))

(:process recover
 :parameters (?b - battery)
 :precondition (>= (delta ?b) 0)
 :effect (and (decrease (delta ?b) (* #t (* (kprime ?b) (delta ?b))))))

(:event batteryDead
 :parameters (?b - battery)
 :precondition (and (switchedOn ?b) (<= (gamma ?b) (* (- 1 (cParam ?b)) (delta ?b))))
 :effect (and (not (switchedOn ?b)) (dead ?b)))
```

A PDDL+ Battery Model

```
(:durative-action use
 :parameters (?b - battery)
 :duration (>= ?duration 0)
 :condition (and (at start (switchedOff ?b)) (over all (switchedOn ?b)))
 :effect (and (at start (not (switchedOff ?b))) (at start (switchedOn ?b))
                        (at end (not (switchedOn ?b))) (at end (switchedOff ?b))))
(:process consume
 :parameters (?b - battery)
 :precondition (switchedOn ?b)
 :effect (and (decrease (gamma ?b) (* #t (load)))
                        (increase (delta ?b) (* #t (/ (load) (cParam ?b))))))

(:process recover
 :parameters (?b - battery)
 :precondition (>= (delta ?b) 0)
 :effect (and (decrease (delta ?b) (* #t (* (kprime ?b) (delta ?b))))))

(:event batteryDead
 :parameters (?b - battery)
 :precondition (and (switchedOn ?b) (<= (gamma ?b) (* (- 1 (cParam ?b)) (delta ?b))))
 :effect (and (not (switchedOn ?b)) (dead ?b)))
```

A PDDL+ Battery Model

```
(:durative-action use
 :parameters (?b - battery)
 :duration (>= ?duration 0)
 :condition (and (at start (switchedOff ?b)) (over all (switchedOn ?b)))
 :effect (and (at start (not (switchedOff ?b))) (at start (switchedOn ?b))
                        (at end (not (switchedOn ?b))) (at end (switchedOff ?b))))
(:process consume
 :parameters (?b - battery)
 :precondition (switchedOn ?b)
 :effect (and (decrease (gamma ?b) (* #t (load)))
                        (increase (delta ?b) (* #t (/ (load) (cParam ?b))))))

(:process recover
 :parameters (?b - battery)
 :precondition (>= (delta ?b) 0)
 :effect (and (decrease (delta ?b) (* #t (* (kprime ?b) (delta ?b))))))

(:event batteryDead
 :parameters (?b - battery)
 :precondition (and (switchedOn ?b) (<= (gamma ?b) (* (- 1 (cParam ?b)) (delta ?b))))
 :effect (and (not (switchedOn ?b)) (dead ?b)))
```

A PDDL+ Battery Model

```
(:durative-action use
 :parameters (?b - battery)
 :duration (>= ?duration 0)
 :condition (and (at start (switchedOff ?b)) (over all (switchedOn ?b)))
 :effect (and (at start (not (switchedOff ?b))) (at start (switchedOn ?b))
              (at end (not (switchedOn ?b))) (at end (switchedOff ?b))))
(:process consume
 :parameters (?b - battery)
 :precondition (switchedOn ?b)
 :effect (and (decrease (gamma ?b) (* #t (load)))
              (increase (delta ?b) (* #t (/ (load) (cParam ?b))))))

(:process recover
 :parameters (?b - battery)
 :precondition (>= (delta ?b) 0)
 :effect (and (decrease (delta ?b) (* #t (* (kprime ?b) (delta ?b))))))

(:event batteryDead
 :parameters (?b - battery)
 :precondition (and (switchedOn ?b) (<= (gamma ?b) (* (- 1 (cParam ?b)) (delta ?b))))
 :effect (and (not (switchedOn ?b)) (dead ?b)))
```

continuous change

A PDDL+ Battery Model

```
(:durative-action use
 :parameters (?b - battery)
 :duration (>= ?duration 0)
 :condition (and (at start (switchedOff ?b)) (over all (switchedOn ?b)))
 :effect (and (at start (not (switchedOff ?b))) (at start (switchedOn ?b))
                        (at end (not (switchedOn ?b))) (at end (switchedOff ?b))))
(:process consume
 :parameters (?b - battery)
 :precondition (switchedOn ?b)
 :effect (and (decrease (gamma ?b) (* #t (load)))
                        (increase (delta ?b) (* #t (/ (load) (cParam ?b))))))

(:process recover
 :parameters (?b - battery)
 :precondition (>= (delta ?b) 0)
 :effect (and (decrease (delta ?b) (* #t (* (kprime ?b) (delta ?b))))))

(:event batteryDead
 :parameters (?b - battery)
 :precondition (and (switchedOn ?b) (<= (gamma ?b) (* (- 1 (cParam ?b)) (delta ?b))))
 :effect (and (not (switchedOn ?b)) (dead ?b)))
```

(switchedOff ?b) is still false,
this battery can no longer be used !

The PDDL+ Battery Planning Problem

```
(define (problem loads1) (:domain kibam)

(:objects battery1 battery2 - battery)

(:init  (satisfactoryService)
        (= (services) 0)
        (switchedOff battery1)
        (= (cParam battery1) 0.166)
        (= (kprime battery1) 0.122)
        (= (gamma battery1) 5.5)
        (= (delta battery1) 0)
        (switchedOff battery2)
        (= (cParam battery2) 0.166)
        (= (kprime battery2) 0.122)
        (= (gamma battery2) 5.5)
        (= (delta battery2) 0)
```

Initial state

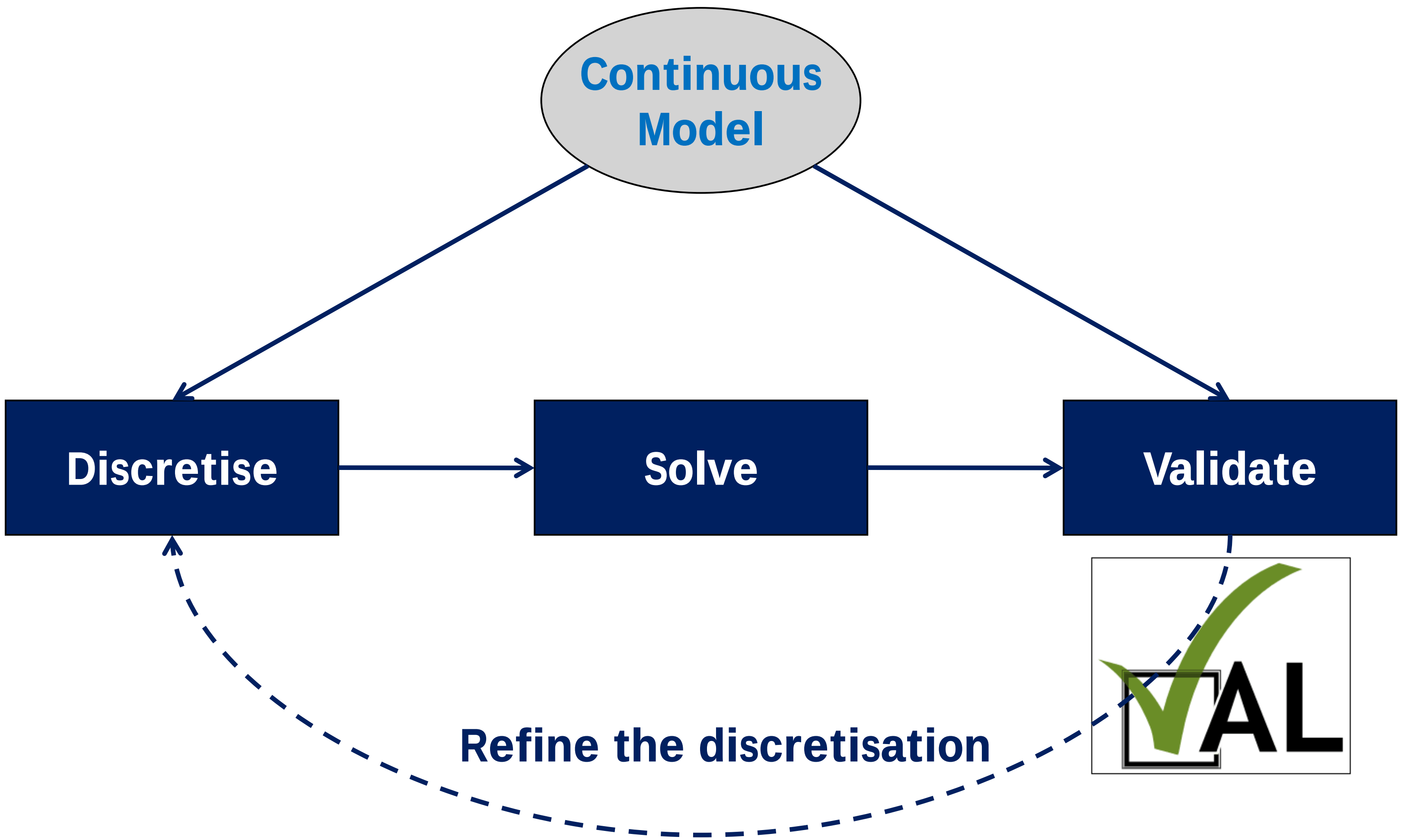
```
(at 0 (= (load) 0.5))
(at 1 (= (load) 0))
(at 2 (= (load) 0.25))
(at 3 (= (load) 0))
(at 4 (= (load) 0.5))
(at 5 (= (load) 0))
(at 6 (= (load) 0.25))
(at 7 (= (load) 0))
(at 8 (= (load) 0.5))
(at 9 (= (load) 0))
(at 10 (= (load) 0.25))
(at 11 (= (load) 0))
(at 12 (= (load) 0.5))
(at 13 (= (load) 0))
(at 14 (= (load) 0.25))
(at 15 (= (load) 0))
(at 16 (= (load) 0.5))
(at 17 (= (load) 0))
(at 17 (allDone))
```

load profile

```
)
```

goal condition (:goal (and (finished))))

The Discretise and Validate Approach



The Monotonicity Property and Planning

- The charge in the battery **monotonically decreases** over time
- The optimal solution is the one that gives the **longest plan**

Heuristic: Plan duration + Available charge

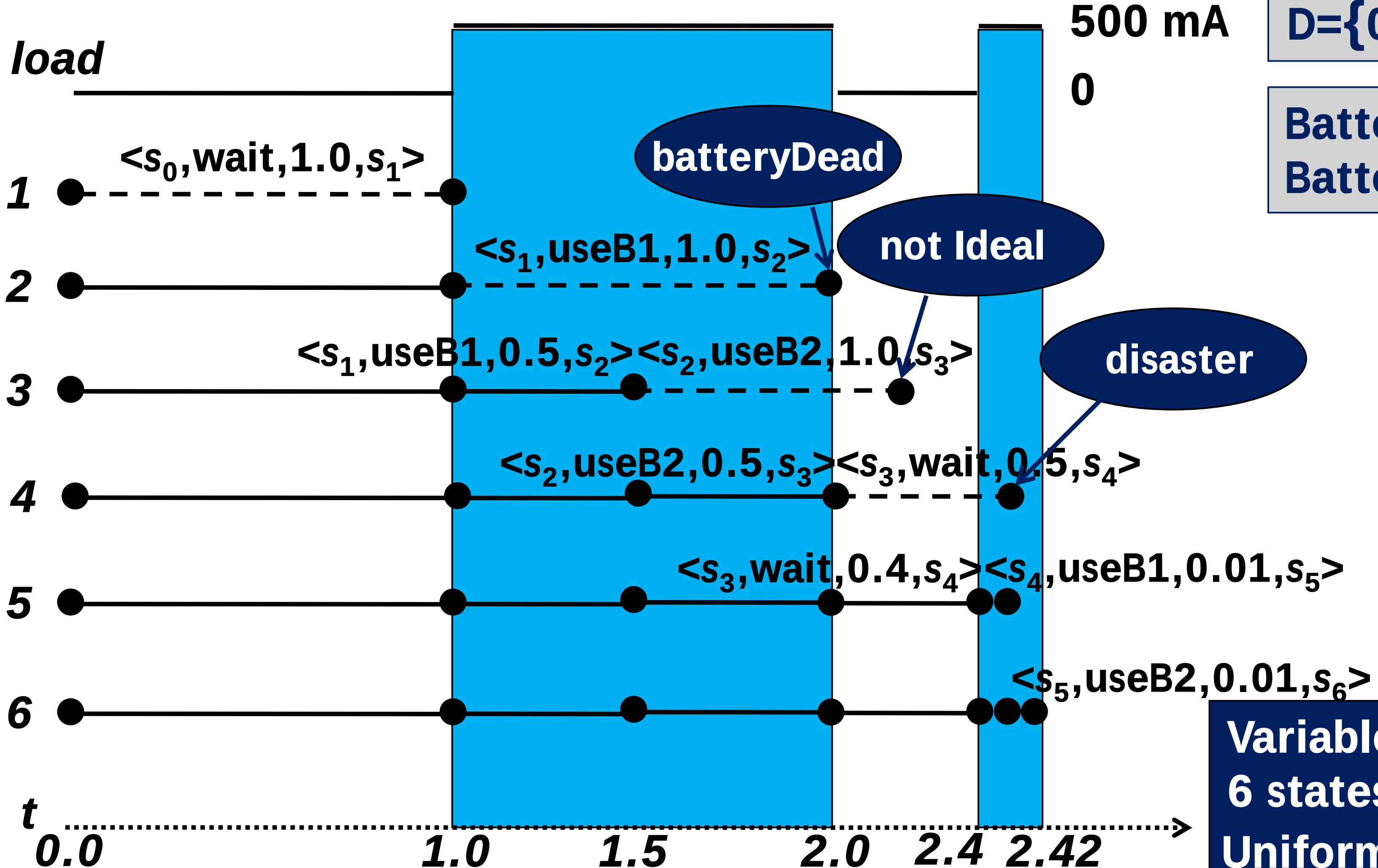
Algorithm 1 Dynamic State Space Search ($\mathcal{P}$)

Input: a planning problem $\mathcal{P} = ((S, s_0, \mathcal{A}, \mathcal{D}, F), G, T)$
Output: a valid plan $\pi*$

$Q \leftarrow (s_0, \texttt{null}, 0)$;
$H \leftarrow s_0$;
if $s_0 \in G$ **then return** $\pi*$;
while $Q \neq \emptyset$ **do**
  $(s_h, a_i, d_k) \leftarrow \text{argmax}_{(s,a,d)\in Q} h(s)$;
  for all $a_j \in \texttt{EnAct}(s_h)$ **do**
    if $a_j \neq a_i$ **then** $\Delta \leftarrow \{d_l \in \mathcal{D} | t(s_h) + d_l \leq T\}$;
    else $\Delta \leftarrow \{d_l \in \mathcal{D} | d_l \leq d_k \wedge t(s_h) + d_l \leq T\}$;
    for all $d_l \in \Delta$ **do**
      $s' \leftarrow F(s_h, a_j, d_l)$;
      if $s' \in G$ **then return** $\pi*$;
      if $s' \notin H$ **then**
        $Q \leftarrow Q \cup (s', a_j, d_l)$;
        $H \leftarrow H \cup s'$;

→ Heuristic
→ Variable time steps (next slide)

Variable Discretisation in BUP



D={0.01,0.4,0.5,1}

Battery B1
Battery B2

Variable discretisation
6 states
Uniform discretisation
> 242 states

Plan Validation through VAL [Fox & Long]

- Automatic simulation tool for executing plan traces.
- Handles all the expressive power of PDDL+ (processes and events).
- Input: a PDDL+ model of a mixed discrete-continuous system, a description of an initial state, a description of a plan
- Output: success or failure, depending on whether the plan describes a valid execution trace through the model.
- VAL report gives the history of the trace sequence, including the metric values of process fluents at each happening on the trace.
- If a trace fails the report gives simple repair advice.

Plan Validation through VAL [Fox & Long]

- VAL produces a detailed report in LaTeX format

14: **Event triggered!**
Activated process (consume battery1)

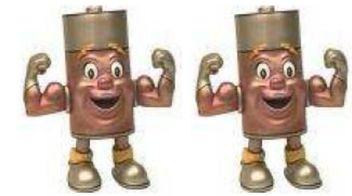
15: Checking Happening... ...OK!

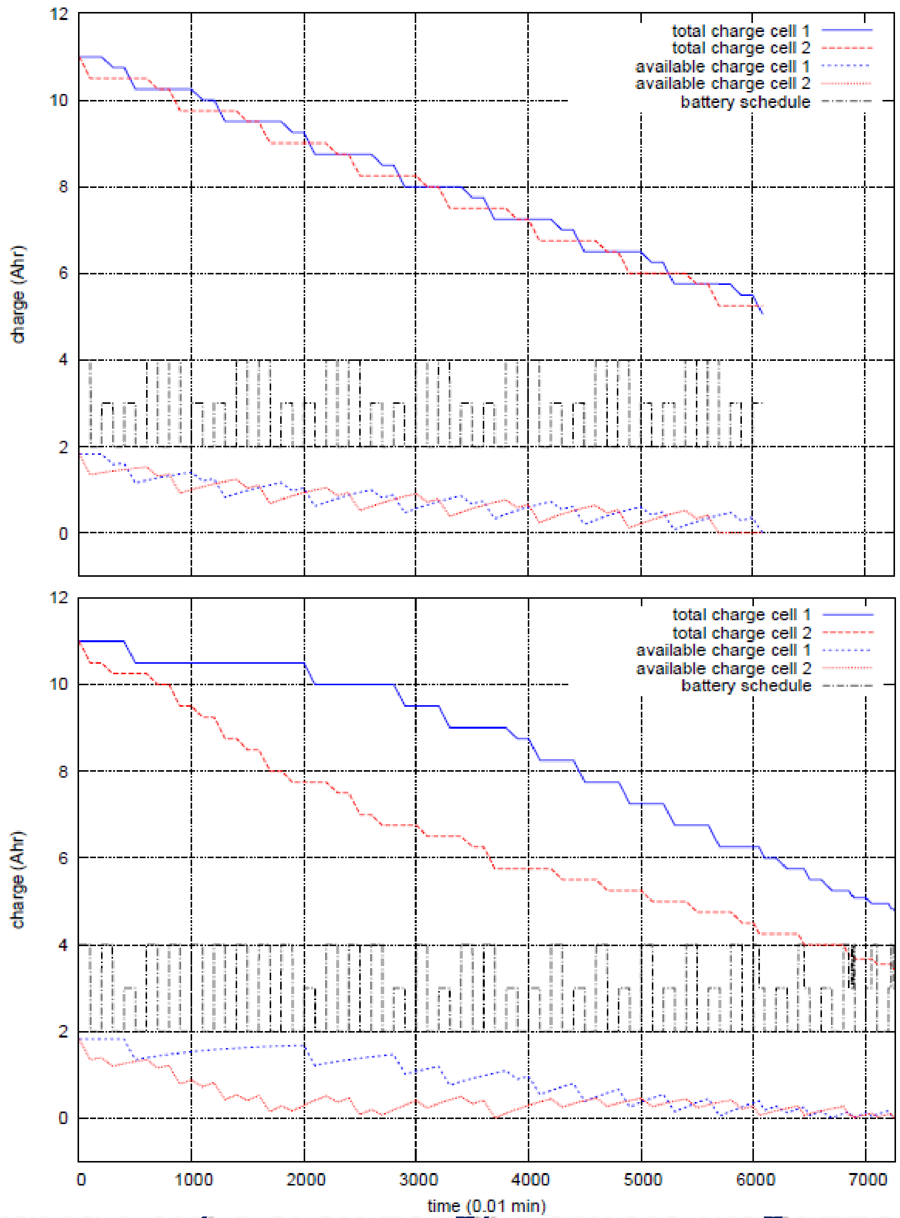
15: Checking Happening... ...OK!
(gamma battery1)$(t) = -0.25t + 4.25$
(delta battery1)$(t) = -8.84646e^{-0.122t} + 12.3445$
(delta battery2)$(t) = 4.03622e^{-0.122t}$
Updating **(gamma battery1)** (4.25) by 4 for continuous update.
Updating **(delta battery1)** (3.498) by 4.51403 for continuous update.
Updating **(delta battery2)** (4.03622) by 3.57265 for continuous update.

15: Checking Happening... ...OK!
Deleting (switchedon battery1)
Adding (switchedoff battery1)
Decreasing **(services)** (1) by 1.
Updating **(load)** (0.25) by 0 assignment.

15: **Event triggered!**
Unactivated process (consume battery1)

Deterministic Solving

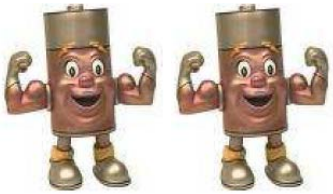




V_{MAX}

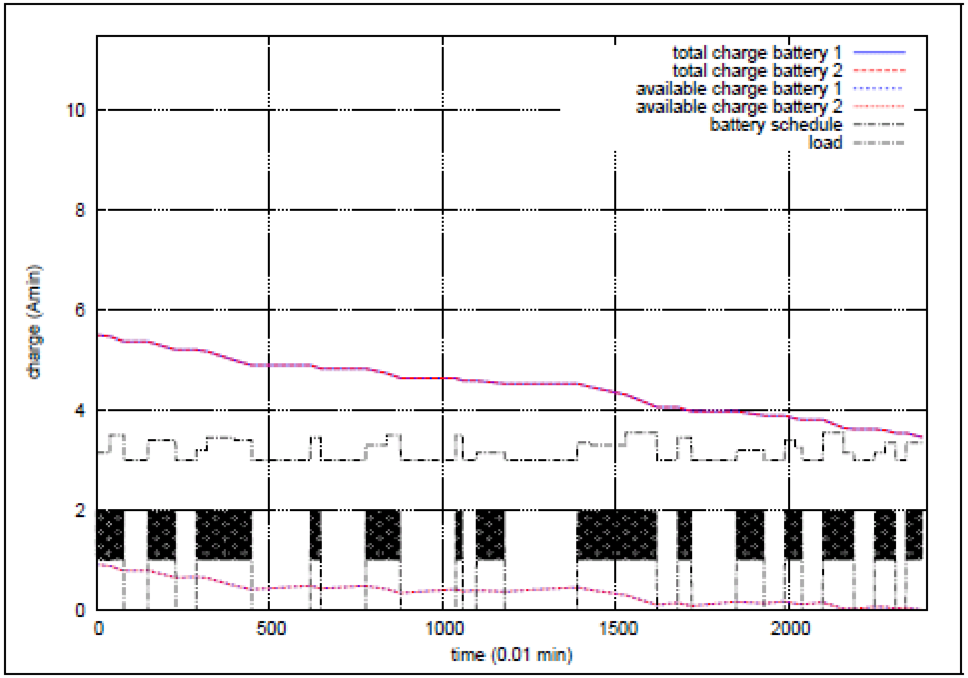
DD-KiBaM

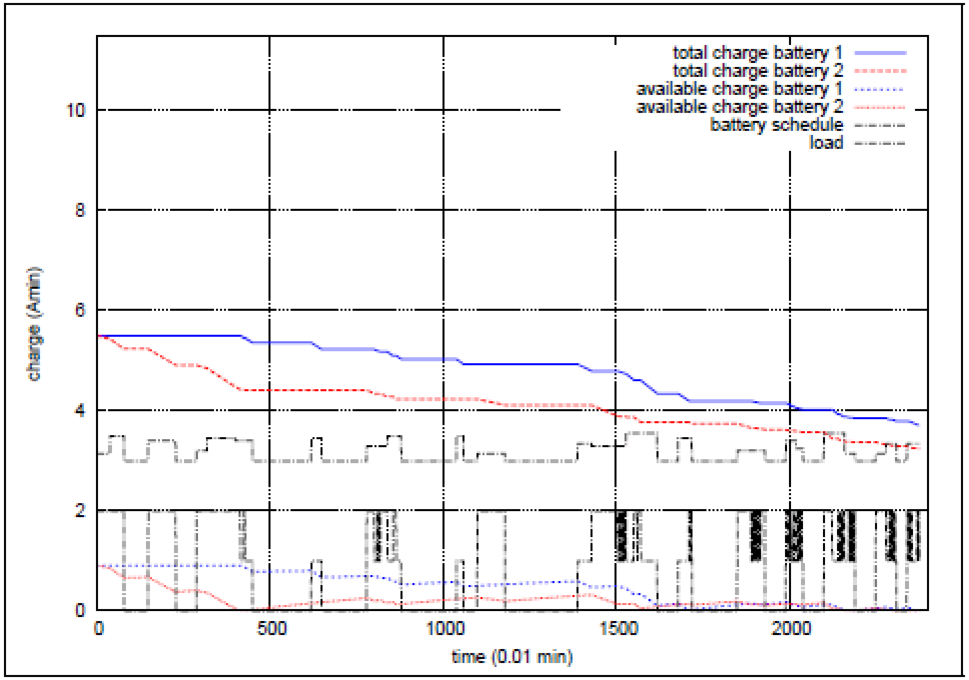
Deterministic Solving



Theoretical Upper Bound

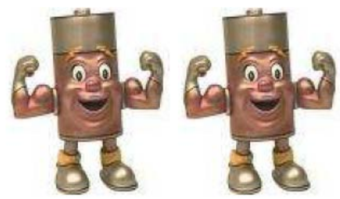
DD-KiBaM





Switches: **1719** **41**

Deterministic Solving



load profile	best-of-two lifetime		UPPAAL-KiBaM lifetime		DD-KiBaM lifetime (visited states)	
	B_1	B_2	B_1	B_2	B_1	B_2
CL_250	12.16	46.92	12.04	N/A	**12.14** (194)	**46.91** (691)
CL_500	4.59	12.16	4.58	N/A	**4.59** (116)	**12.14** (194)
CL_alt	7.03	21.26	6.48	N/A	**7.03** (136)	**21.2** (350)
ILs_250	44.79	132.8	40.80	N/A	**44.76** (552)	**132.7** (1068)
ILs_500	10.82	44.79	10.48	N/A	**10.8** (131)	**44.76** (552)
ILs_alt	16.95	72.75	16.91	N/A	**16.92** (159)	**72.55** (599)
IL*l*_250	84.91	216.9	78.96	N/A	**84.88** (488)	**216.8** (1123)
IL*l*_500	21.86	84.91	18.68	N/A	**21.85** (173)	**84.88** (488)

Theoretical upper bound

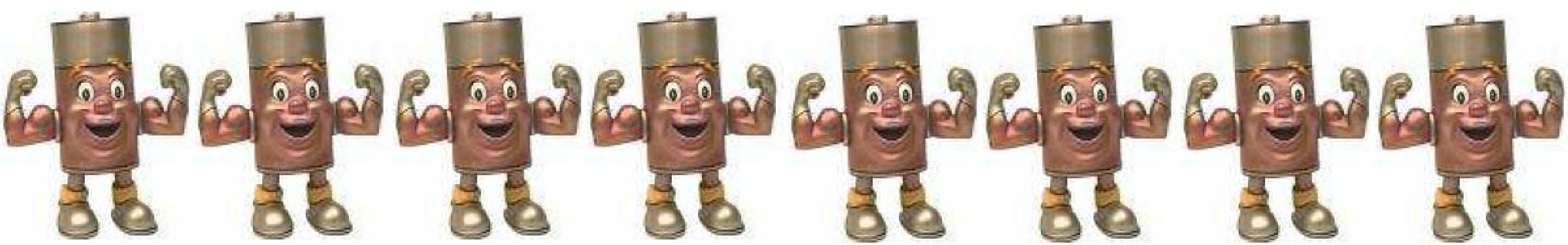
Jongerden et al. 2009

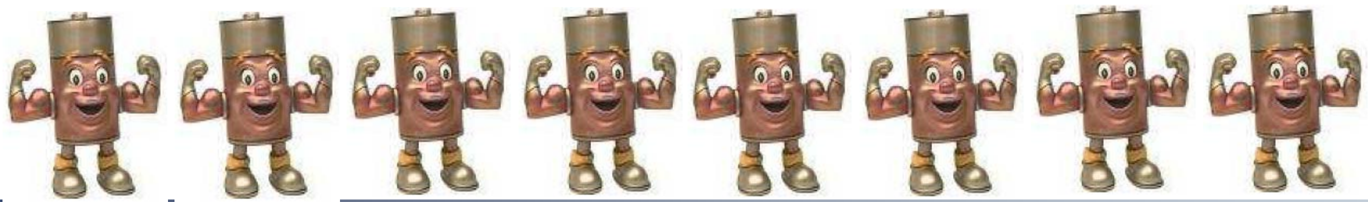
All lifetimes are over 99% of the upper bound

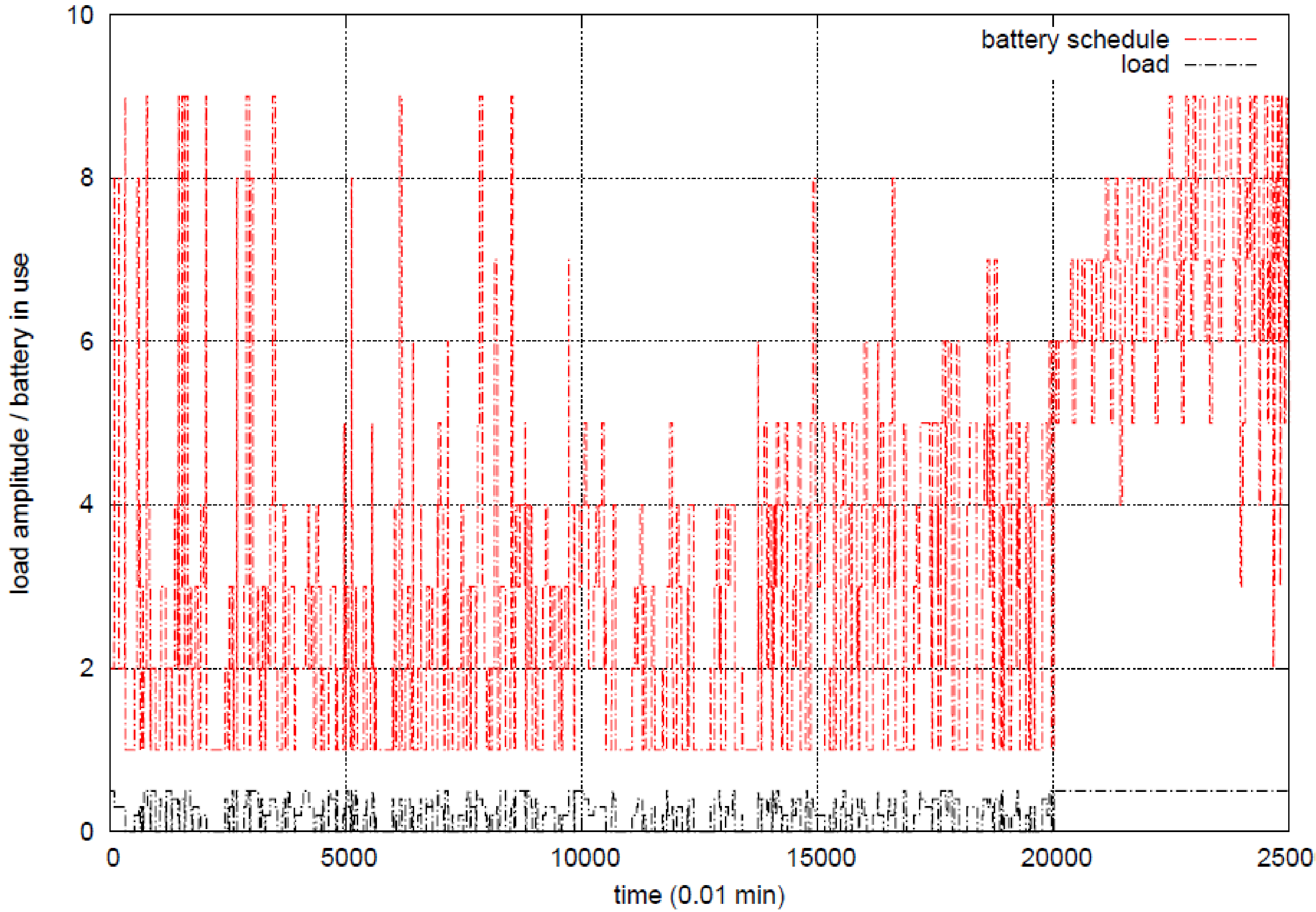
Deterministic Solving

State Space Size: $\mathbf{2^{27} * 10^{80}}$

- **27** predicates
- **16** real variables rounded to **.00001**







21,320 states
16 seconds

99.4% efficiency

VALidated

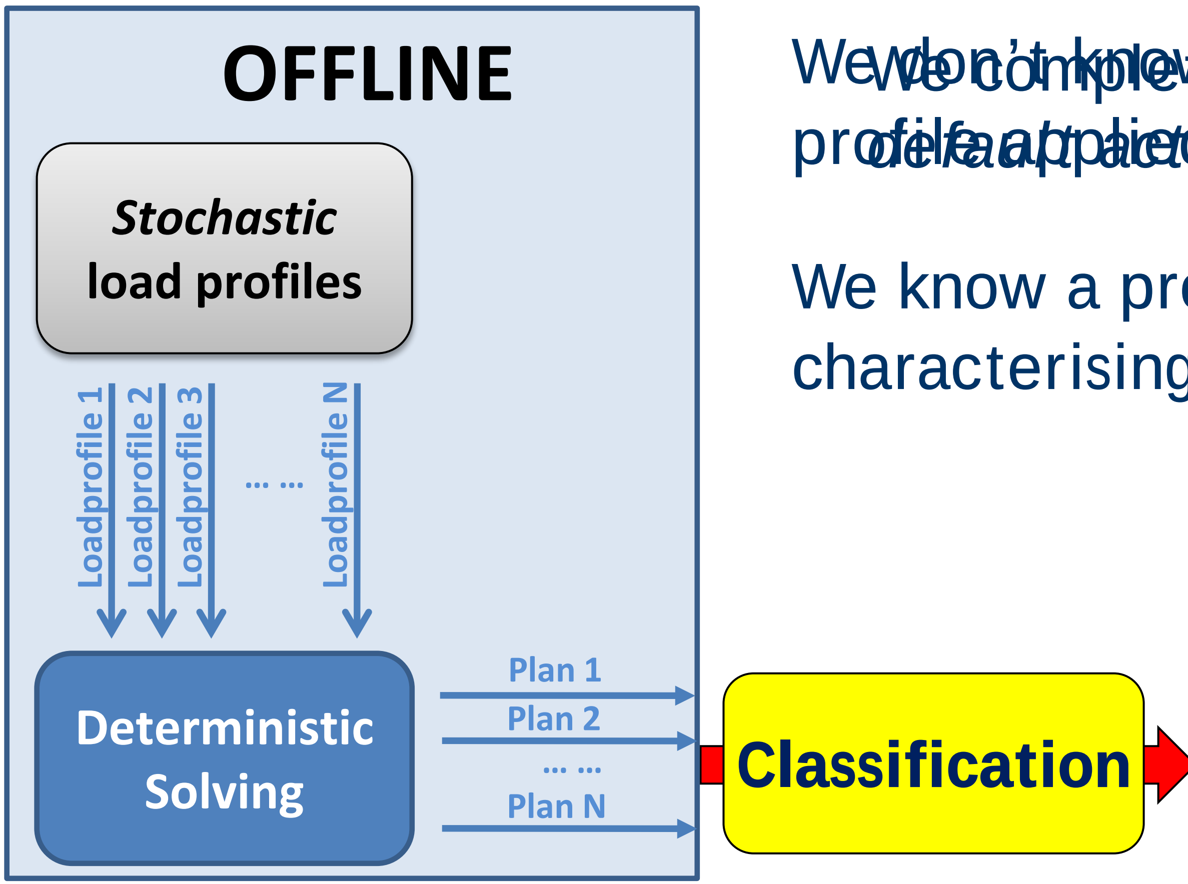
From Plans to Policies

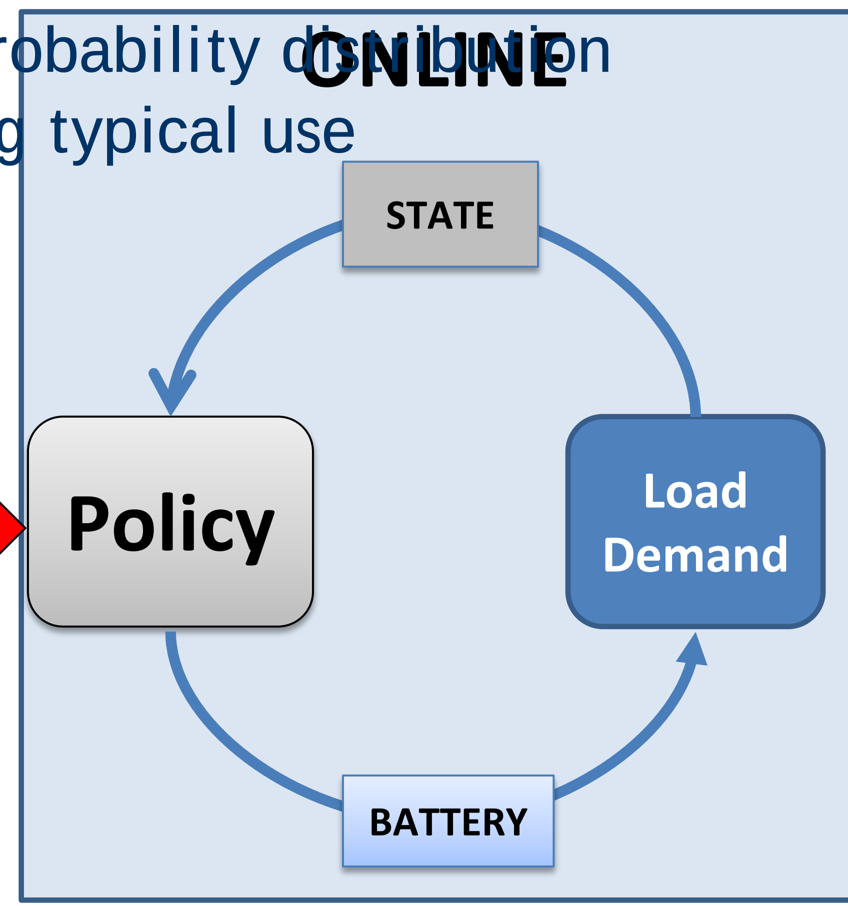
From Plans to Policies

We don't know in advance the load profile applied to the batteries

We complete the policy with a *default action* (best-*a*)

We know a probability distribution characterising typical use



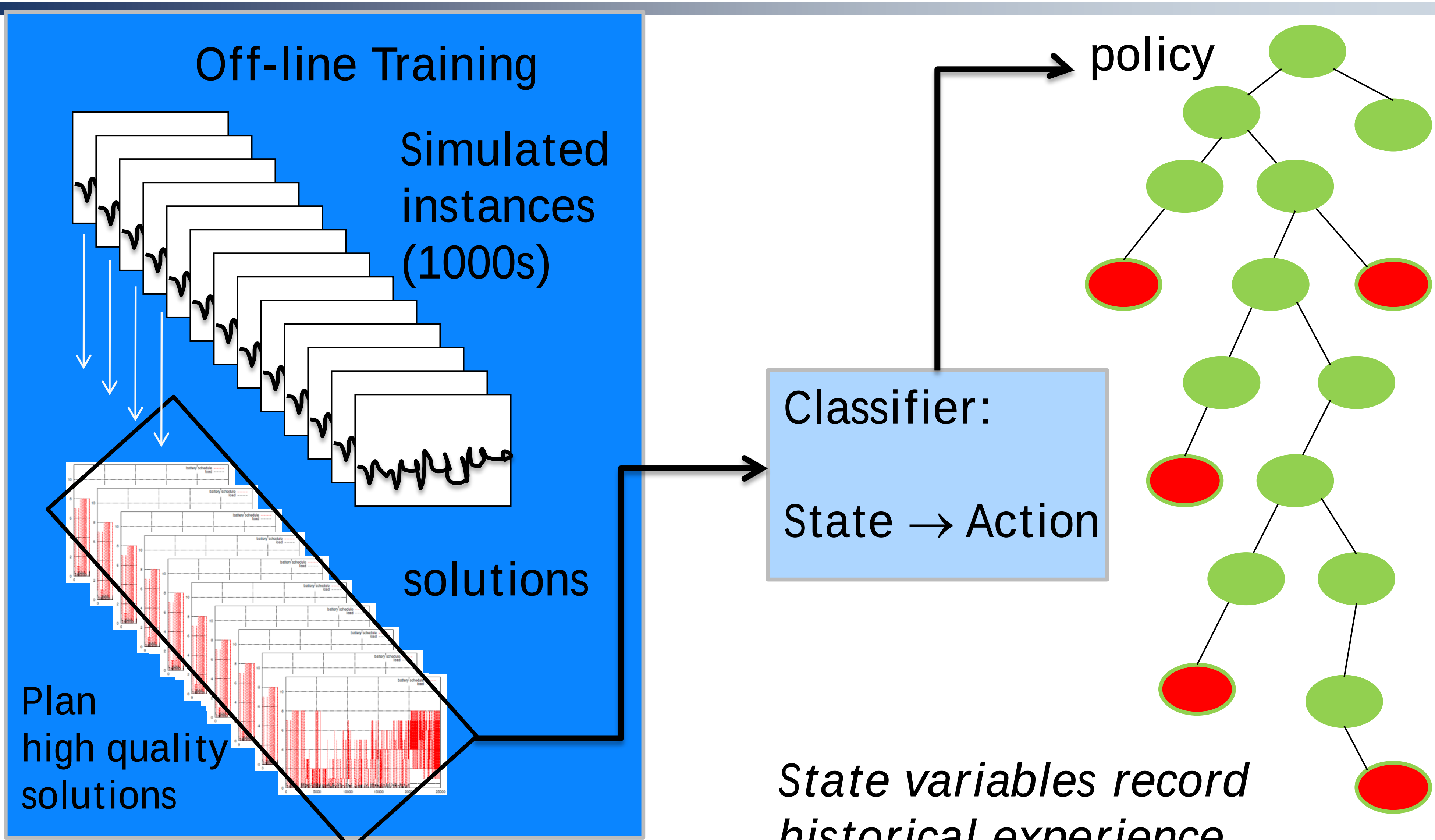


Policy Learning through Classification

- **WEKA** framework (University of Waikato)
- **J48** classifier – based on machine learning algorithm C4.5
- Stochastic load profiles based on probability distributions for:
  - load amplitude *l*
  - load/idle period duration *d*
  - load frequency *f*

- The output is a *decision tree* (leaves = the battery to be used)

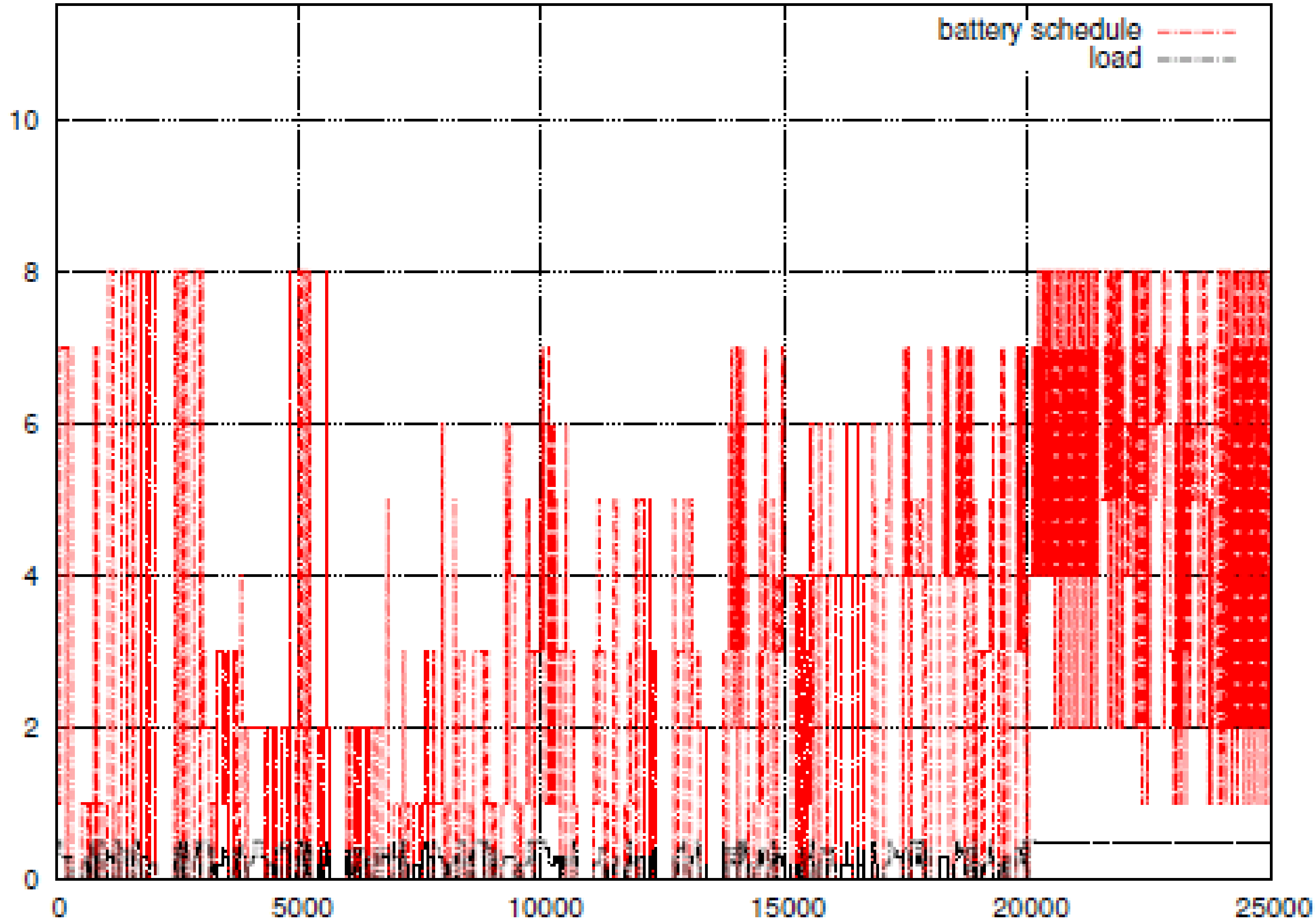
Policy Learning through Classification



State variables record historical experience

Results

Best-of-n: 16640 **DD-Plan: 424** **DD-Policy: 852**



Results

- 4 probability distributions with different average load amplitude
- L in [100, 250, 500, 750] mA

load profile	best-of-8 time$_{(\sigma)}$	best-of-8 $sw_{(\sigma)}$	DD-Policy time$_{(\sigma)}$	DD-Policy $sw_{(\sigma)}$
R100	792.6$_{(15.5)}$	71383$_{(1379)}$	**786.2**$_{(15.4)}$	**1667**$_{(161)}$
R250	369.8$_{(1.91)}$	28952$_{(853)}$	**366.7**$_{(2.02)}$	**1518**$_{(143)}$
R500	226.7$_{(2.13)}$	14671$_{(512)}$	**224.6**$_{(2.27)}$	**987**$_{(122)}$
R750	188.3$_{(0.8)}$	11519$_{(463)}$	**186.4**$_{(0.7)}$	**302**$_{(33)}$

Conclusions

- Effective solution to a multiple battery management problem
- Our solution achieves better than 99% efficiency (existing solutions achieve no better than 95%)
- In many applications even a small margin can be of considerable added value
- The approach is scalable and effective

Future work:

- To deal with rechargeable batteries
- To use the approach on different domains

Thanks!

Maria Derek Dan