BEACONS: An end-to-end Compiler Framework for Predicting and Utilizing Dynamic Loop Characteristics

Girish Mururu*, Sharjeel Khan*, Bodhi Chatterjee*, Chao Chen,
Chris Porter, Ada Gavrilovska, Santosh Pande

*equal contribution

Georgia Tech

Main Contributions

Compiler-Directed Predictive Analysis

Estimation of an application's dynamic resource requirements just-in-time, in terms of loop characteristics

Loop Trip Count Analysis

Novel loop analysis which divides loops into distinct classes and devises specific mechanisms to estimate their dynamic trip counts.

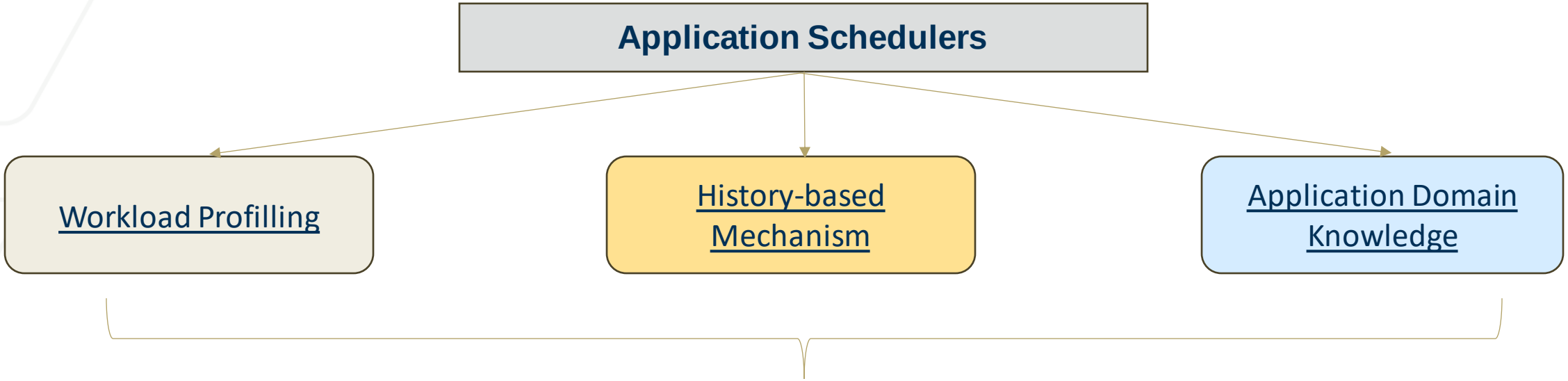
Proactive Workload Scheduling

Throughput-oriented scheduling mechanism that aggregates the dynamic information to allocate resources to the processes

*We present **Beacons Framework**, an end-to-end compiler and scheduling framework, that estimates dynamic loop characteristics, encapsulates them in compiler-instrumented beacons in an application, and broadcasts them during application runtime, for proactive workload scheduling*

Georgia Tech

Resource Allocation in High-Performance Computing (HPC) Clusters



These approaches are agnostic to the fact that *workloads are input-dependent* and can *fluctuate at different program phases*.

Detection and Reaction lag

The goal is to *minimize the overlap between resource-heavy phases* of applications, and *maximize the concurrency in non-resource heavy phases dynamically*

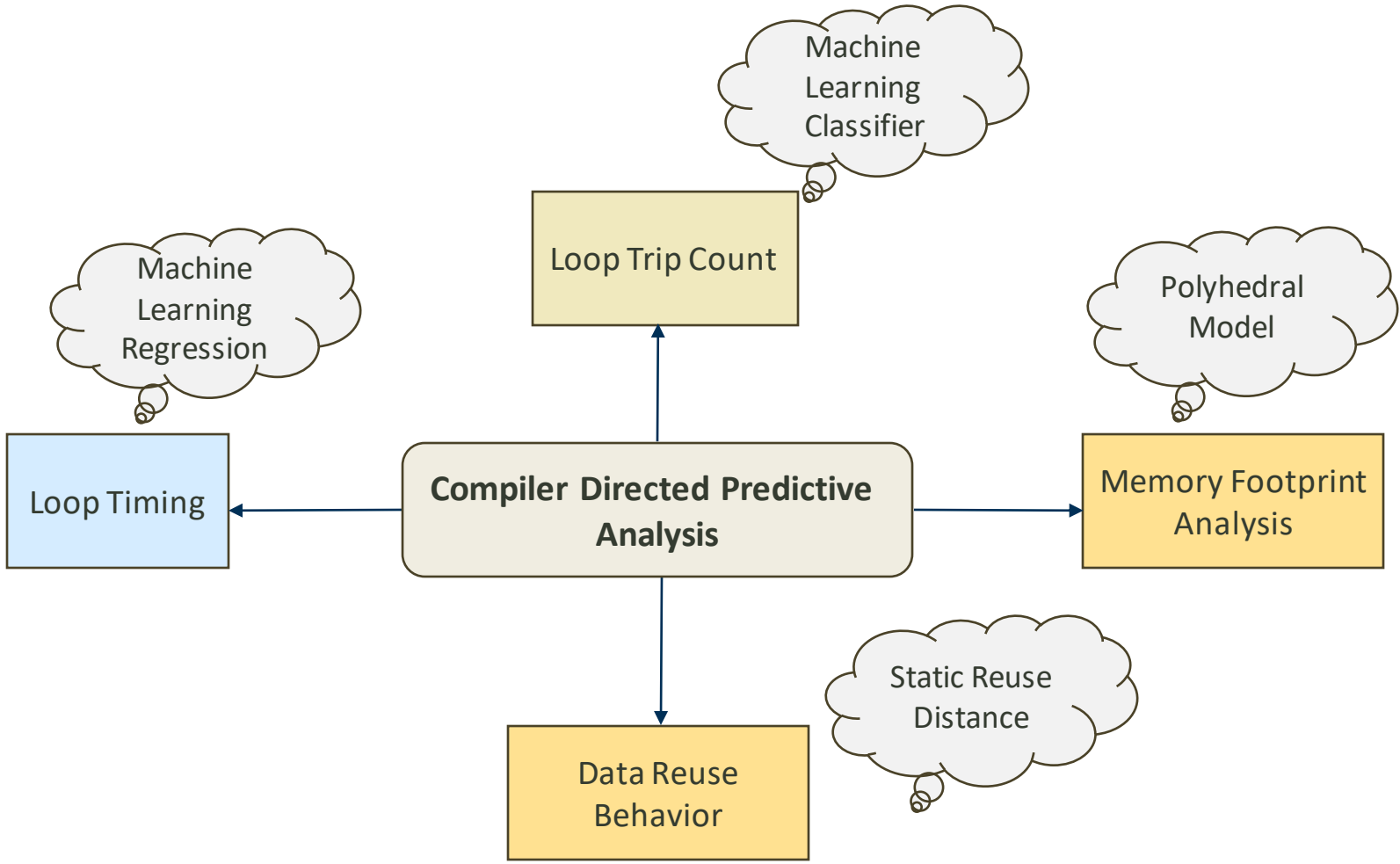
Georgia Tech

Application Resource Requirement = Dynamic Loop Characteristics

Dynamic Resource Requirements are estimated just-in-time in terms of *loop characteristics*

The characteristics are statically computed in lightweight manner through closed-form formulae and machine learning models

At runtime, these models are instantiated and the information is broadcasted to the scheduler for proactive workload scheduling

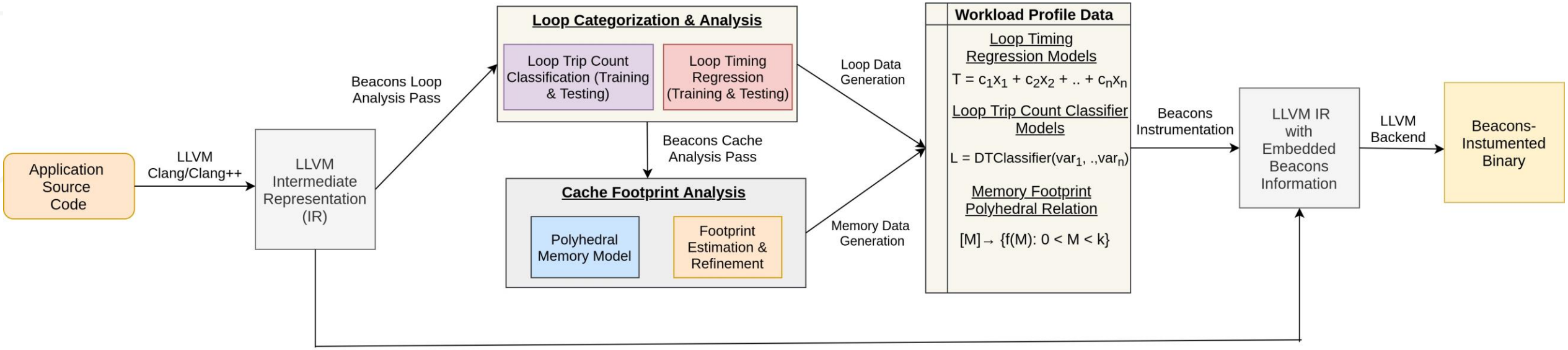


*The characteristics are encapsulated in "**Beacons**" (specialized library markers), and statically instrumented in the application code*

Georgia Tech

Beacons Framework: Compile-time & Run-time Cooperative Scheduling

- *Beacons Compilation Component* consists of multiple compiler passes that estimate various loop characteristics
- The source code is statically instrumented with *prediction models*, which are instantiated at the runtime for information broadcast



- This framework is also responsible for 'hoisting' the beacon calls to the outer-most loop preheaders
- The information broadcast is obtained by designing a commnunication library that uses shared memory to relay the loop information

Georgia Tech

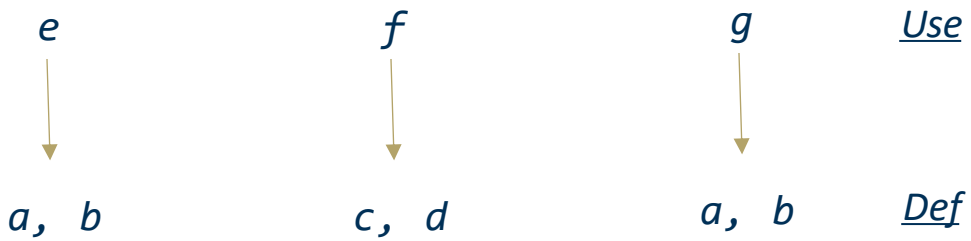
Tackling Irregular Loops: "Backslicing" Critical Variables

- Program variables that dictate the loop iteration conditions are analyzed, and are *decomposed* into critical variables by traversing their def-use chains backwards to the loop pre-header.

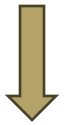
Multi-Exit Loop Nest

```
int a = initVar(10),
    b = initVar(20);
    c = initVar(30);
    d = initVar(40);
do  {
    /*loop body*/
    e += a * b;
    f += c / d;
    if (e == f)
        break;
    g = a + b;
} while (g);
```

Bound and Branch Predicate Variables



Critical Backsliced Variables



Loop nests with complex control flows and indeterminate loop bounds involving *structure fields*, *aliased variables* and function calls can be tackled in this manner

- Loops Nests are categorized into different classes based on their bounds and exit conditions, which helps in determining critical variable sets

Georgia Tech

Loop Categorization Scheme

- To determine the set of *critical variables*, loop nests are divided into four distinct categories:

NBNE

```
for (j = 0; j < _PB_M ; j ++)
    /* loop body */
    for (i = 0; i < _PB_N ; i ++)
        /* loop body */
```

Normally-Bounded Normal Exit

- Trip-Count could be statically estimated
- Critical Variables = ϕ

IBNE

```
while (a[i] < a[j]) {
  /*Loop Body*/
}
```

Irregularly-Bounded Normal Exit

- Trip-Count **cannot** be statically estimated
- Critical Variables = loop-bounds

NBME

```
for (j = 0; j < parameters ; j ++) {
  i++;
  if (i == ( ssize_t ) * num_args )
    break ;
}
```

Normally-Bounded Multi Exit

- Trip-Count **cannot** be statically estimated
- Critical Variables = branch predicates

IBME

```
for (p= classp ; p !=0; p= p -> next_val)
    if (p -> is_cons && !REG_P (p -> exp)) {
      /* loop body */
      break ;
}
```

Irregularly-Bounded Multi Exit

- Trip-Count **cannot** be statically estimated
- Critical Variables = branch predicates **U** loop-bounds

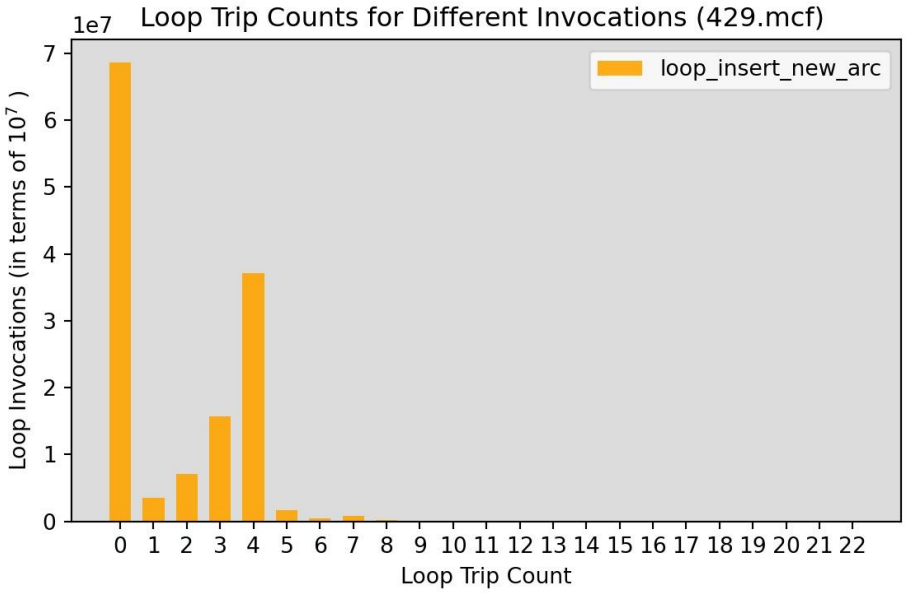
- A loop nest is considered to be "*irregularly bounded*", if any of its bounds are *non-numerical* , or if it's *unbounded* and contains a *branch predicate expression* for the loop termination
- A loop nest is considered to be "multi exit", if it contains multiple *termination conditions*, i.e it contains exiting edges in the control-flow graph which originate from a loop basic block that neither *dominates* nor *post-dominates* the loop body
- *Based on this categorization, over 55% of loop nest in modern workloads are irregular*

Georgia Tech

Loop Trip Count Estimation: Classification Problem

- The distribution of loop trip counts follows a *discrete*, *non-negative, integral* distribution.
- Intuitively, each integer in this loop trip count domain can be thought of as a **class/category**.
- **The goal here is to learn a function that can project the set of critical variables to a loop trip count value.**

```
1 void insert_new_arc(/*function args*/){
2  /*function body*/
3  while(pos-1 && red_cost>new[pos/2-1].
       flow){
4     /*loop body*/
5  }
6 }
```



- This learned function extrapolates the loop trip count distribution, and thus, unlike loop-profiling, is also able to map unseen inputs to precise loop trip counts.
- Scalability of labels is not an issue since our experiments have shown that unique trip counts exhibited by loop nest < 30

Georgia Tech

Loop Timing Analysis: Regression Problem

- **Loop-Timing** is defined as the time taken for executing an entire loop-nest. It helps us to determine how long a particular phase will last.

- **Theorem.** *For a normalized loop nest L with n inner-nested loops with individual upper-bounds* $\{U_1, U_2, .., U_n\}$ *the timing* $\mathcal{T}_c$ *is given by the linear equation:*

$$T_c = U^T C = c_0 + c_1 u_1 + c_2 u_2 + ... + c_n u_n$$

where $C = \{c_0, c_1, ..c_n\}$ are learnable parameters

$U = \{u_0, u_1, .., u_n\}$ is the feature vector

$u_i = \prod_{k=1}^{i} U_k$ represents the individual feature

- During runtime, the actual loop-bounds are plugged into this equation to generate the phase-time.

- The timing model results in an *upper-bound* on loop timing, resulting in conversative scheduling decisions

Georgia Tech

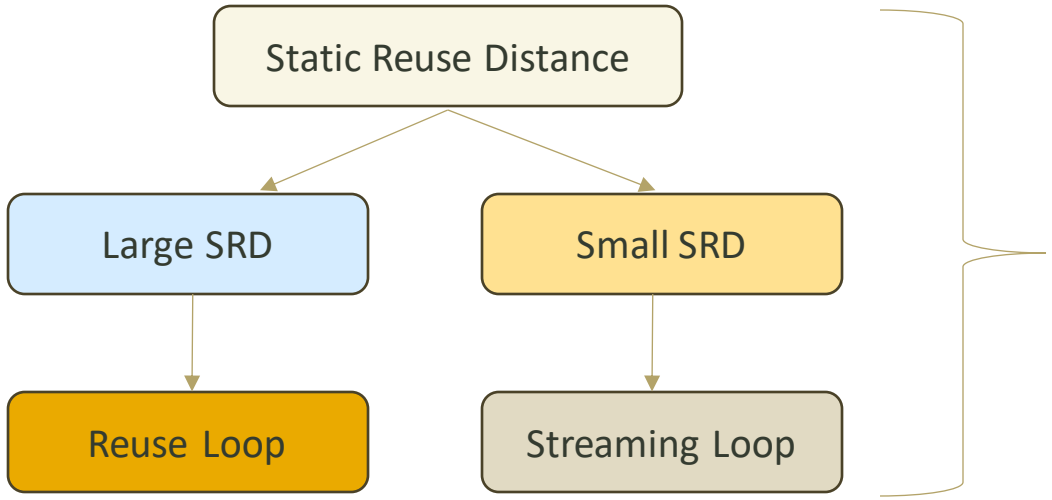
Loop Memory Footprint & Data Reuse: Polyhedral Model

- **Memory footprint** determines the amount of cache that will be utilized by a loop-nest during an execution phase.
- Polyhedral analysis generates memory footprint equations of the form:

$$[X] \rightarrow \{m(X) : 0 < X < N\}$$

m(X) represents the dynamic memory accesses
N is the loop trip count

- For non-affine loops, the memory footprint is enhanced with the precise loop trip count prediction.
- To determine the amount of cache required by a loop-nest for maximizing locality, Beacons Framework uses *Static Reuse Distance* (SRD)



For *large reuse distances*, large cache resources must be allocated so that the reused data will be found in the cache.

For *small reuse distances*, data which is reused only after a couple of iterations will be found in the cache.

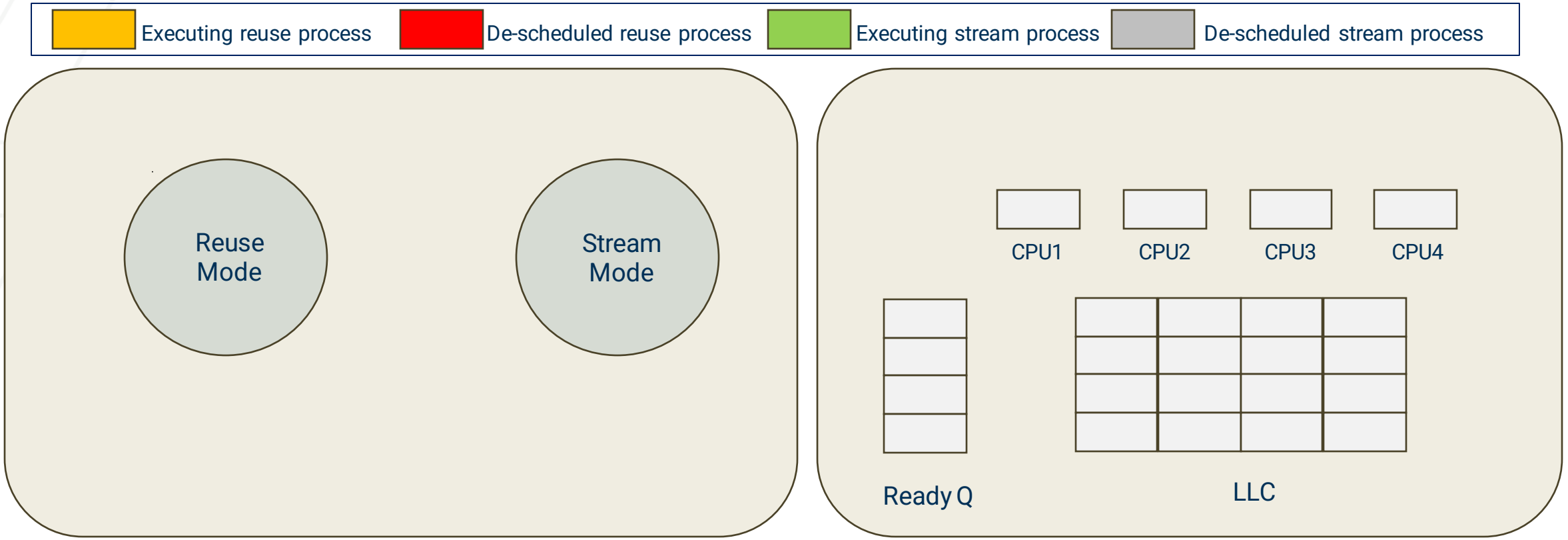
Georgia Tech

Proactive Scheduling: Promoting Concurrency & Minimizing Resource Contention

- Scheduler optimally shares the Last-Level Cache (LLC) among scheduled processes by changing between the two modes

- **Reuse Mode**:
  - A reuse beacon for a loop will lead to a check if all current processes fit in the cache. If not, the process will be put in waiting queue until a spots opens for it.
  - A streaming beacon for a loop is suspended until no more reuse processes are active.
  - When no more reuse processes or **90%** of processes are streaming, we switch to streaming mode.
- **Streaming Mode**:
  - Streaming Mode executes as many streaming processes as possible without exceeding the memory bandwidth
  - When we have many reuse processes in the waiting, we switch back to reuse mode.

Working of The Beacon Scheduler

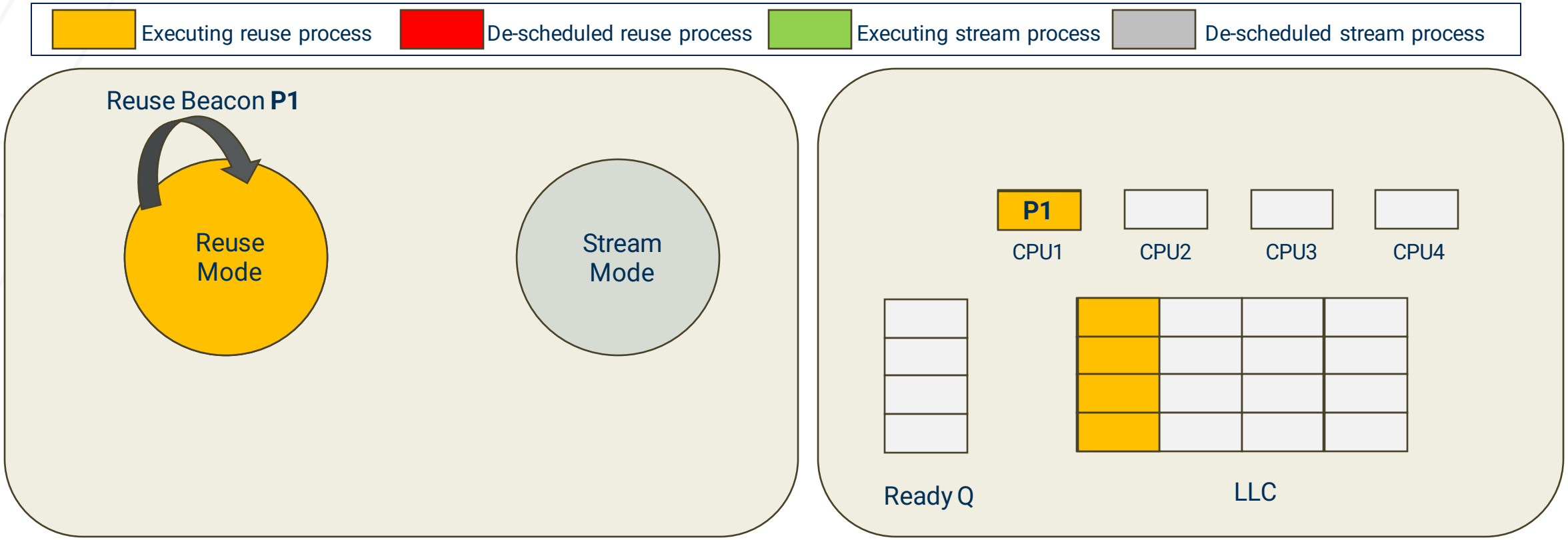


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

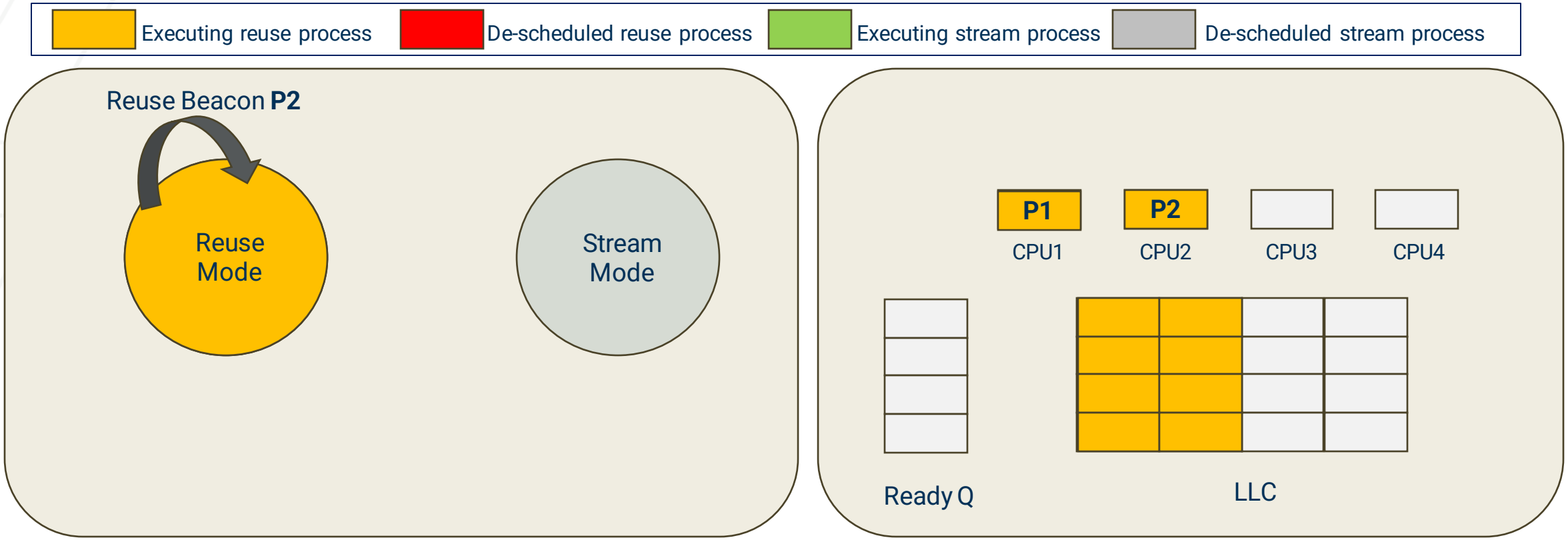
Working of The Beacon Scheduler



Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Working of The Beacon Scheduler

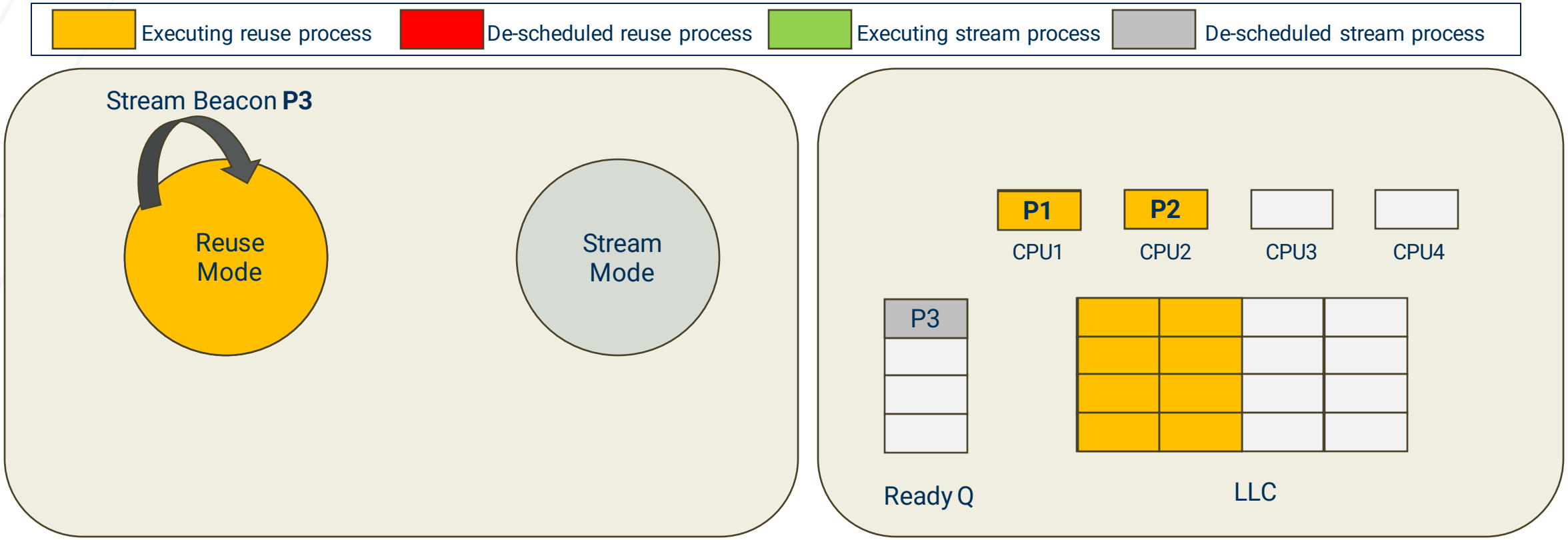


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

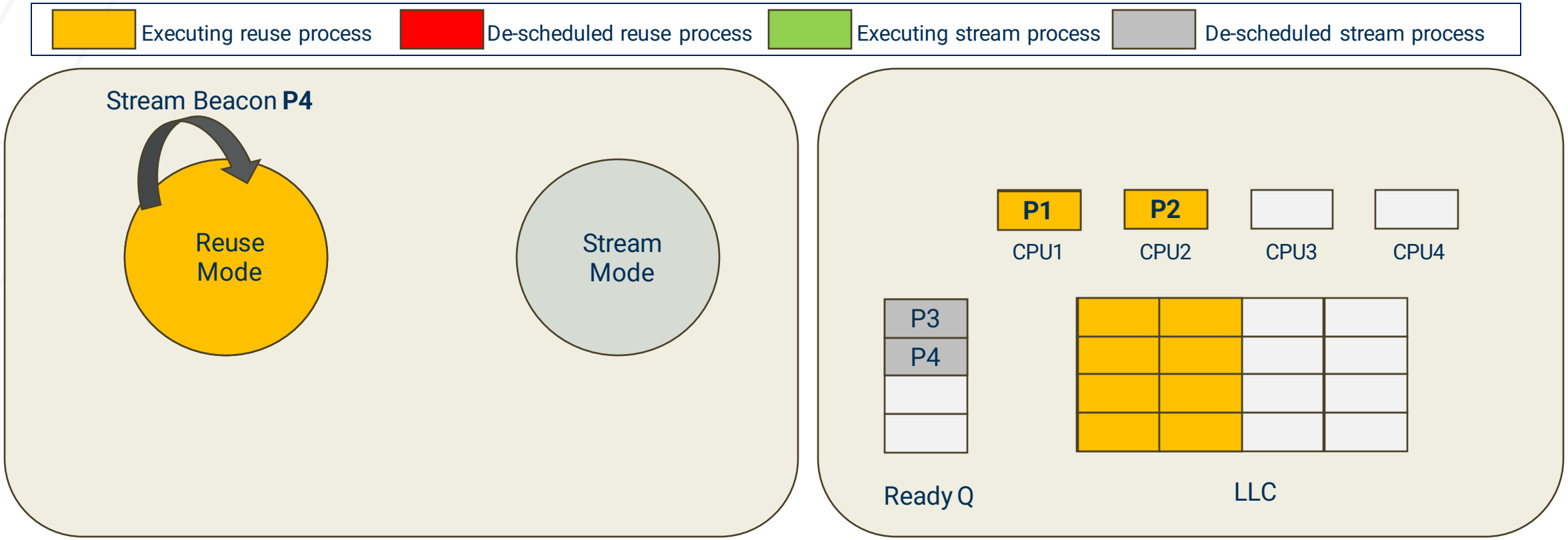


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

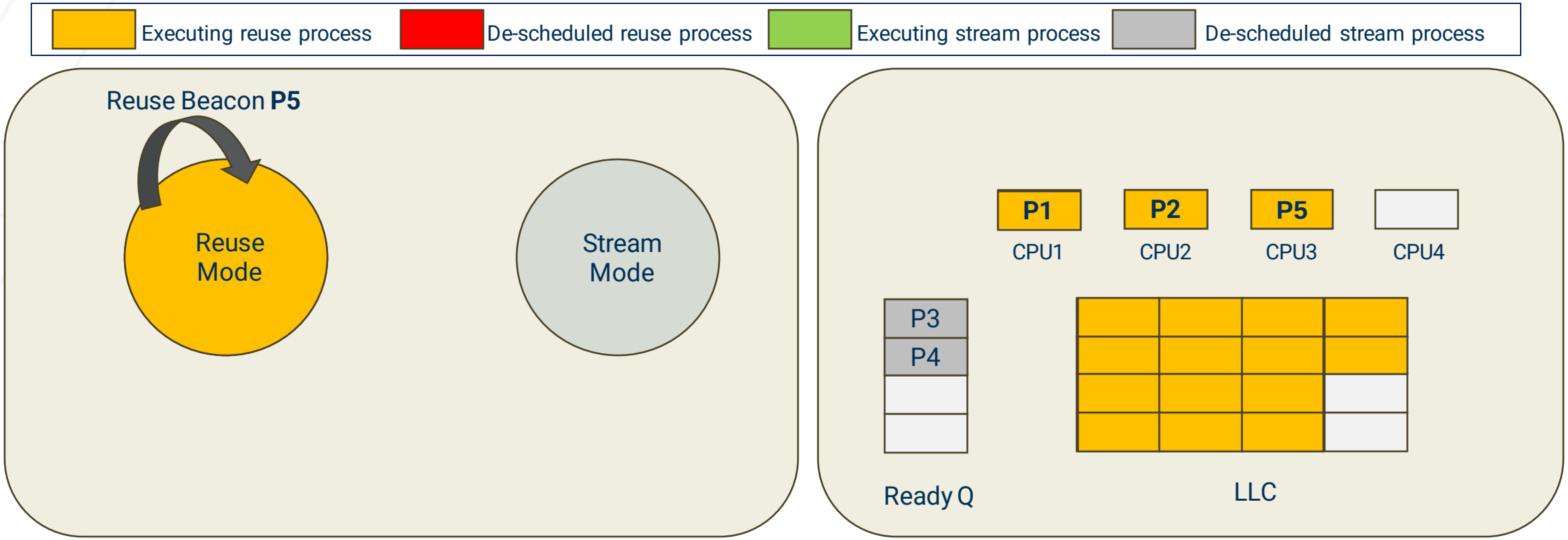


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

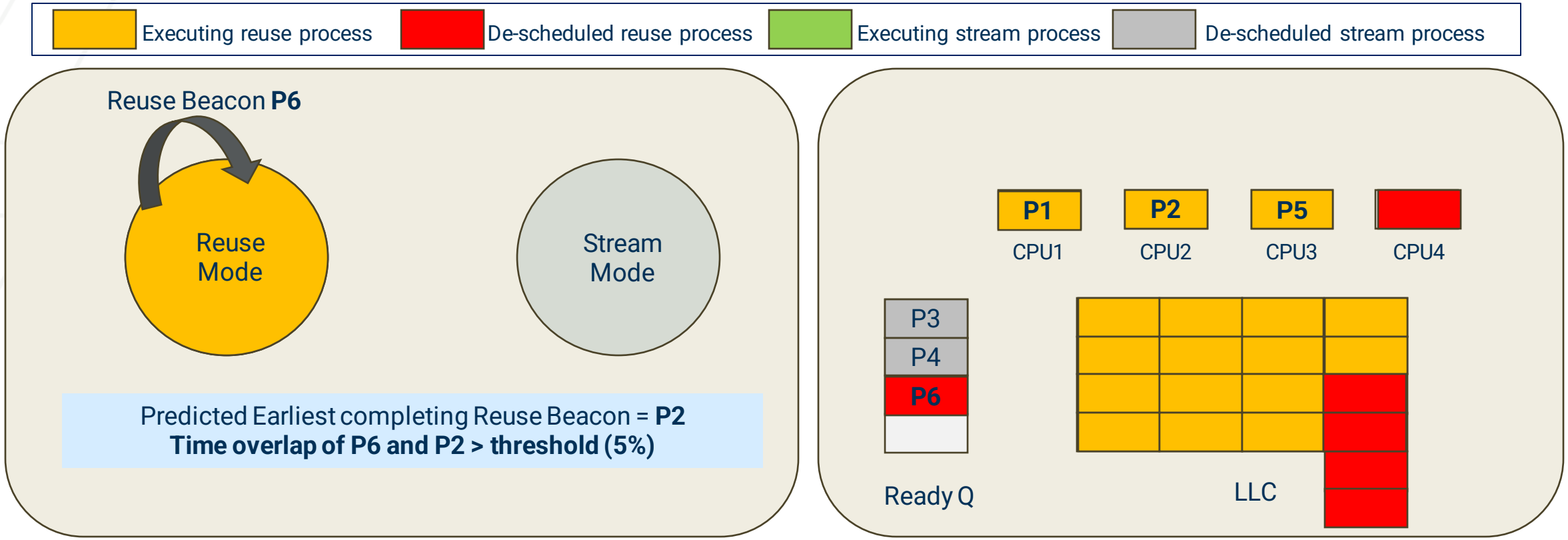


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

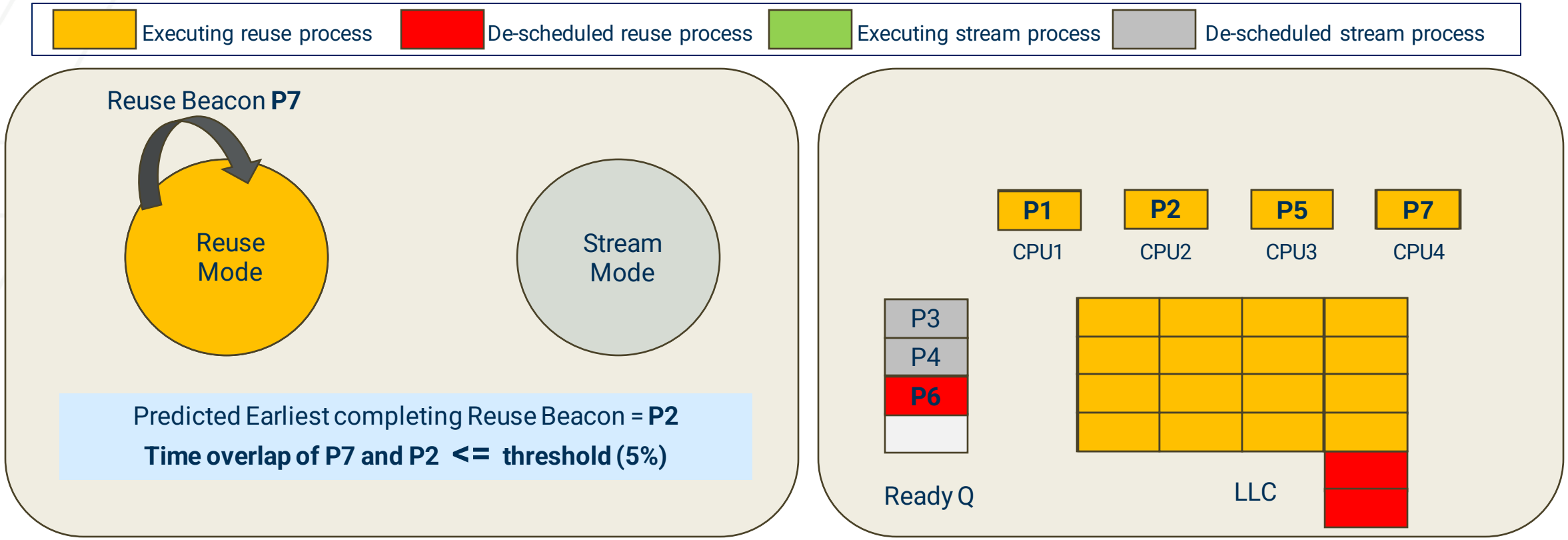
Working of The Beacon Scheduler



Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Working of The Beacon Scheduler

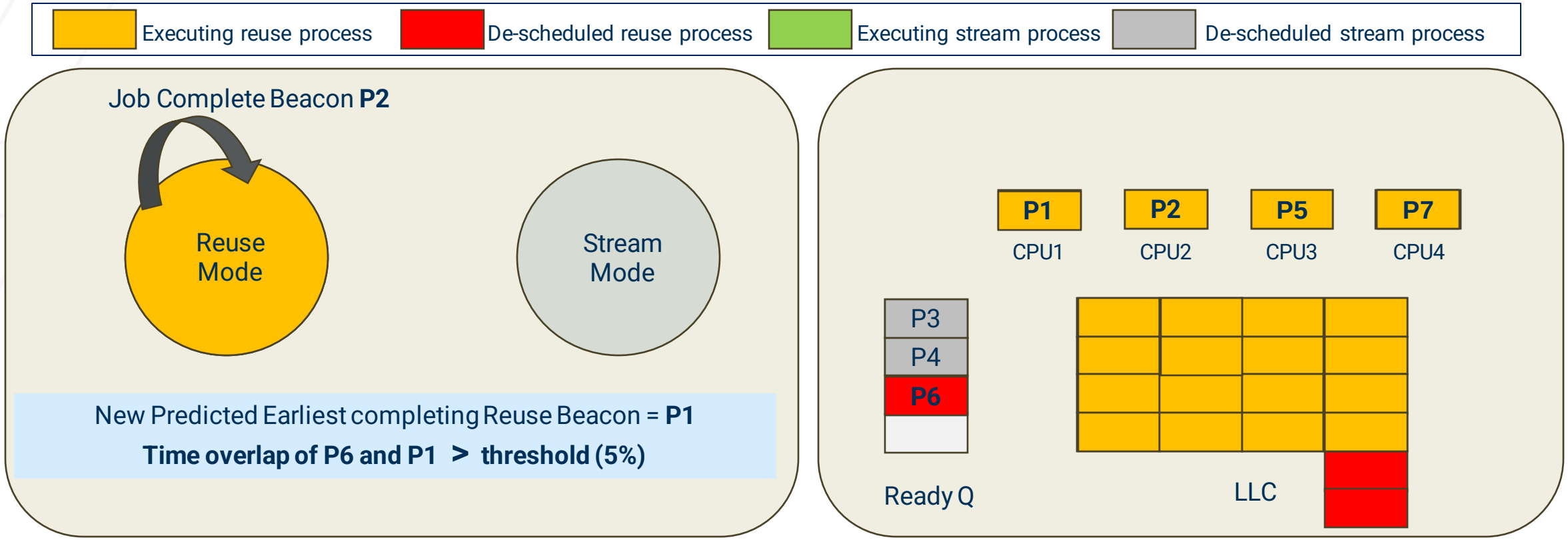


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

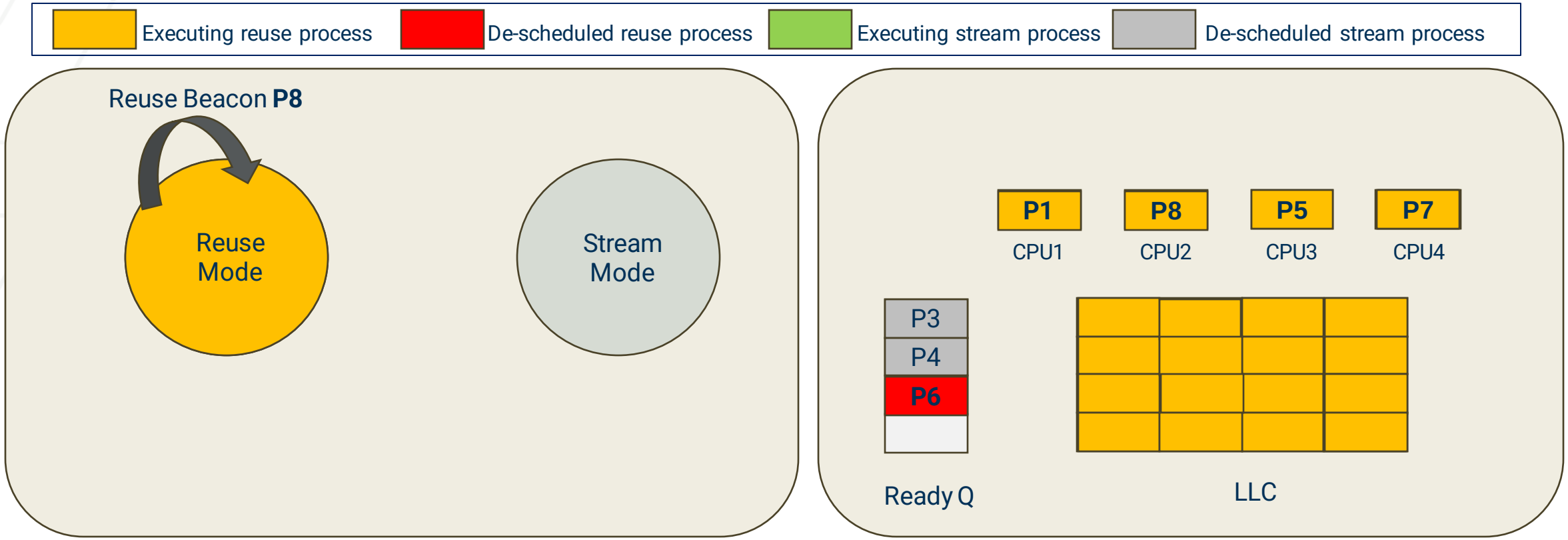


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

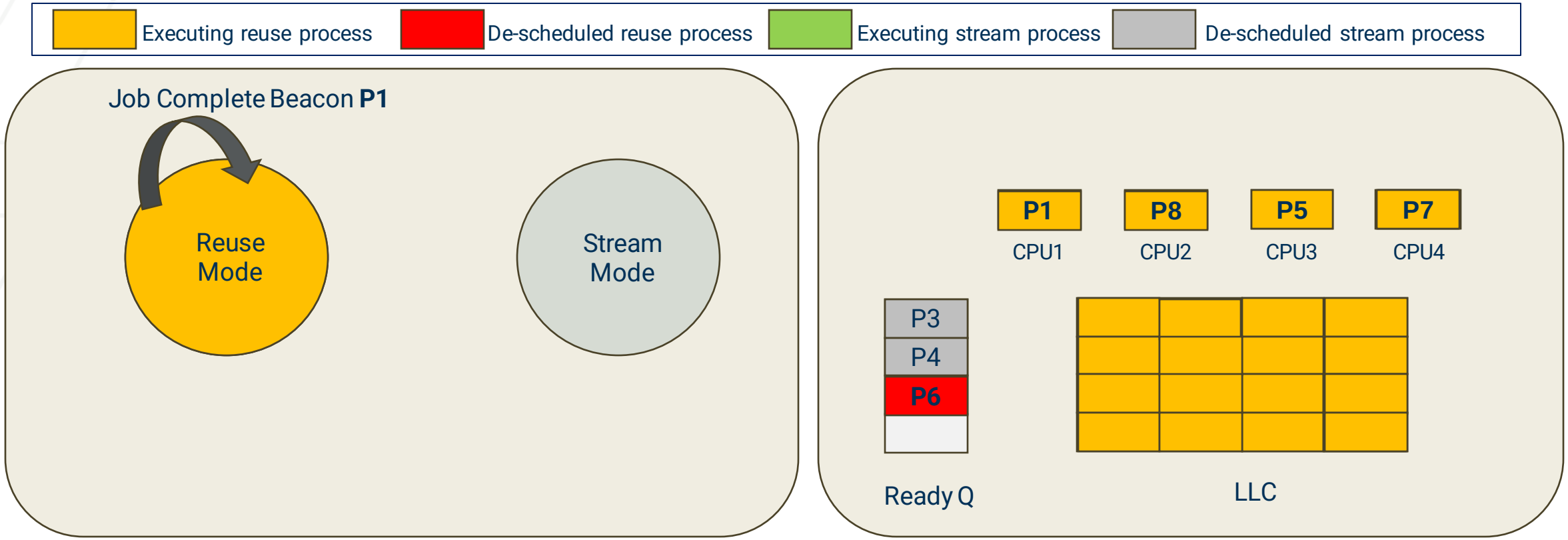


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

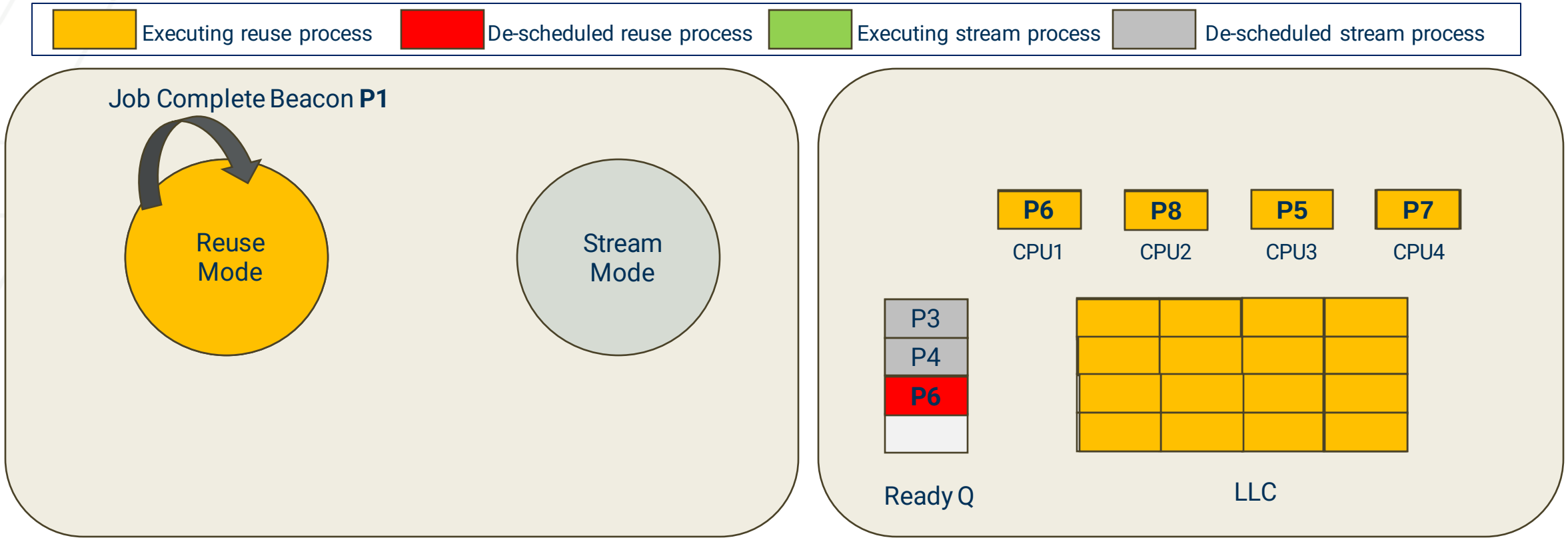


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

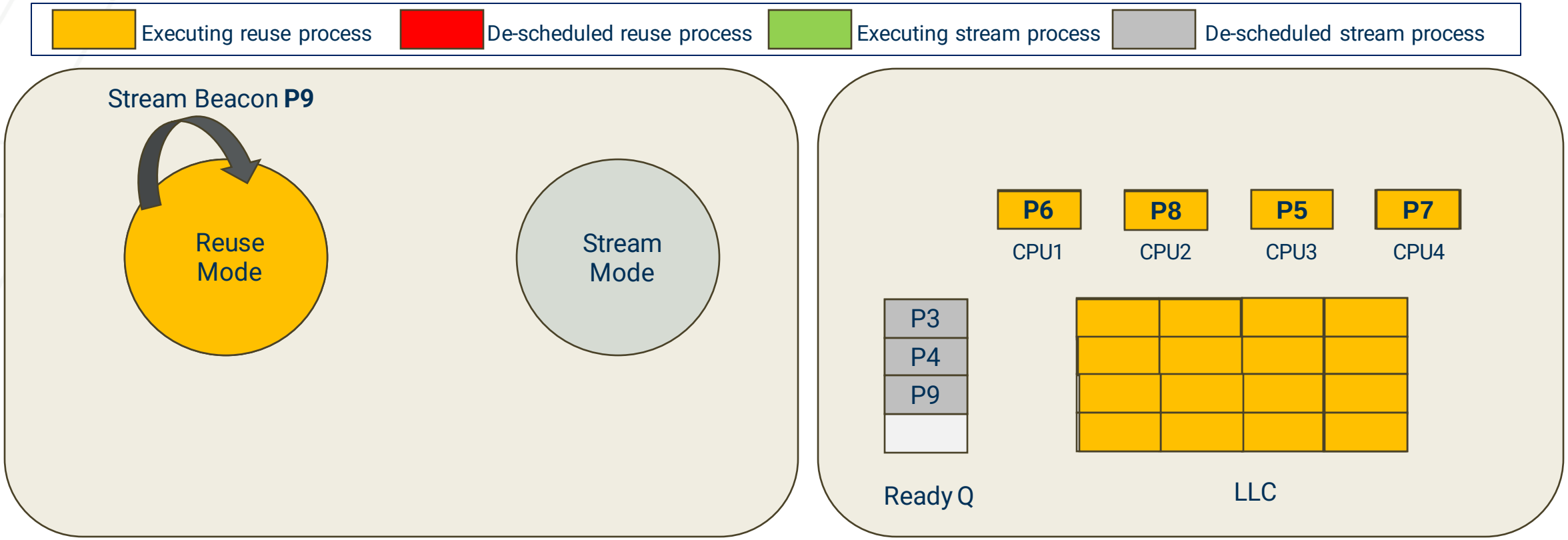


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

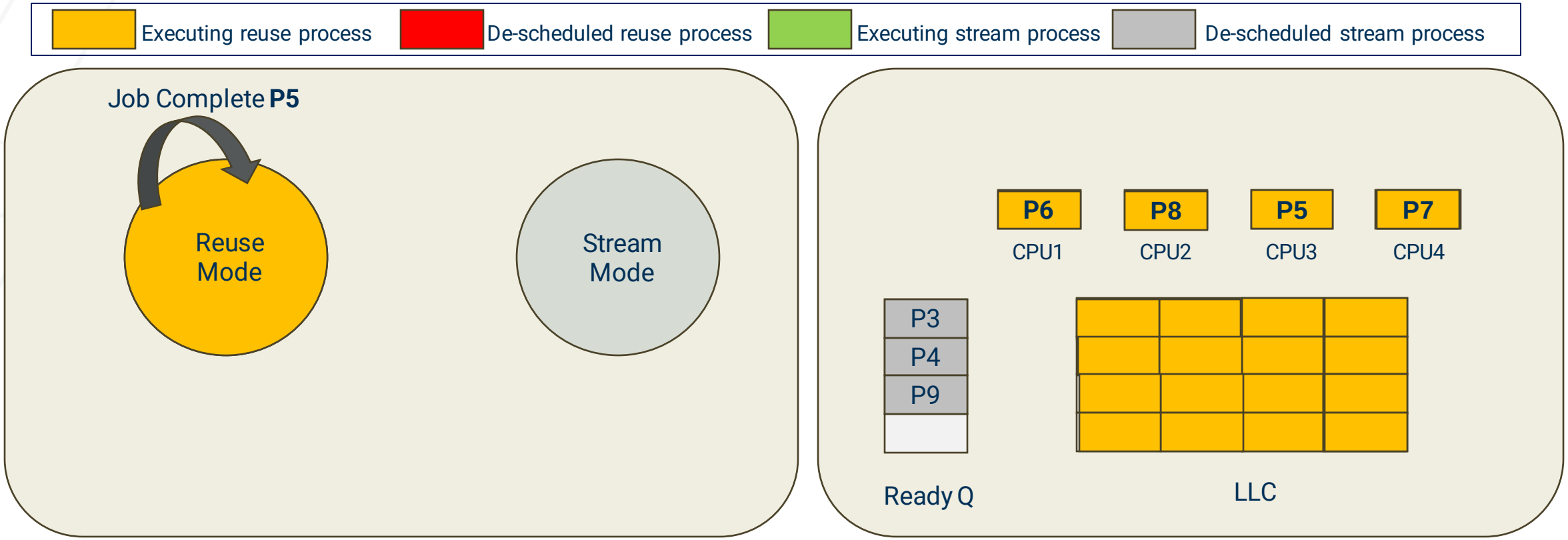
Working of The Beacon Scheduler



Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

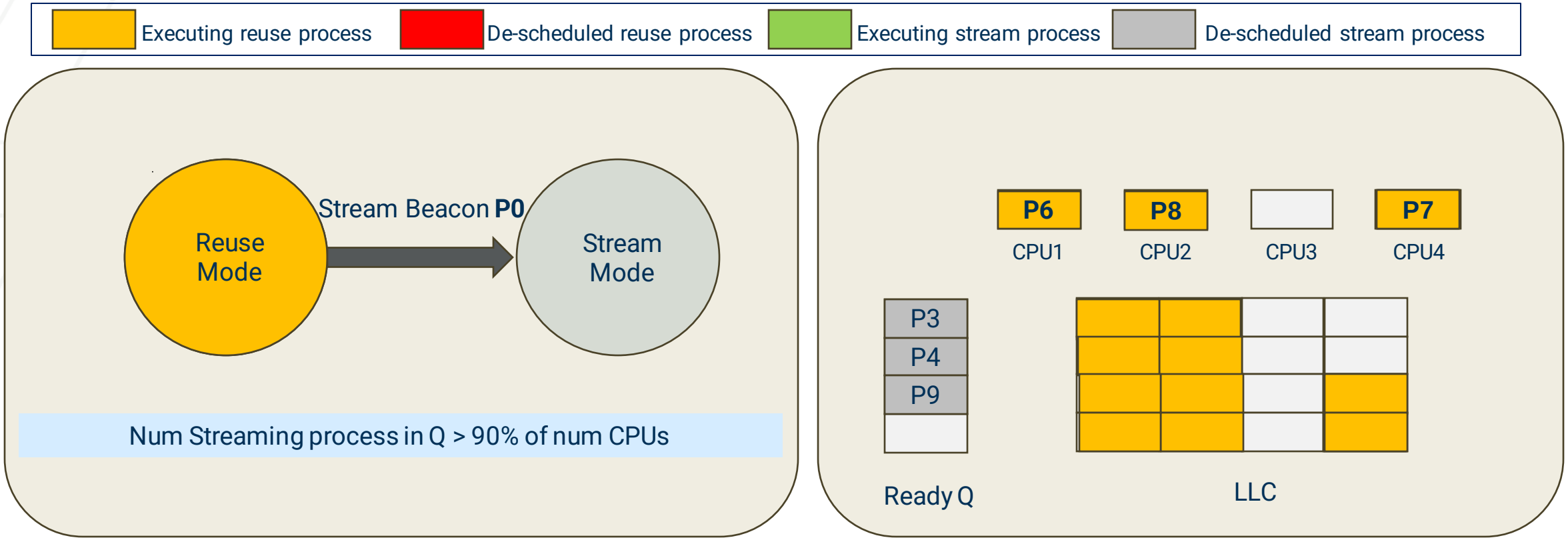
Working of The Beacon Scheduler



Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

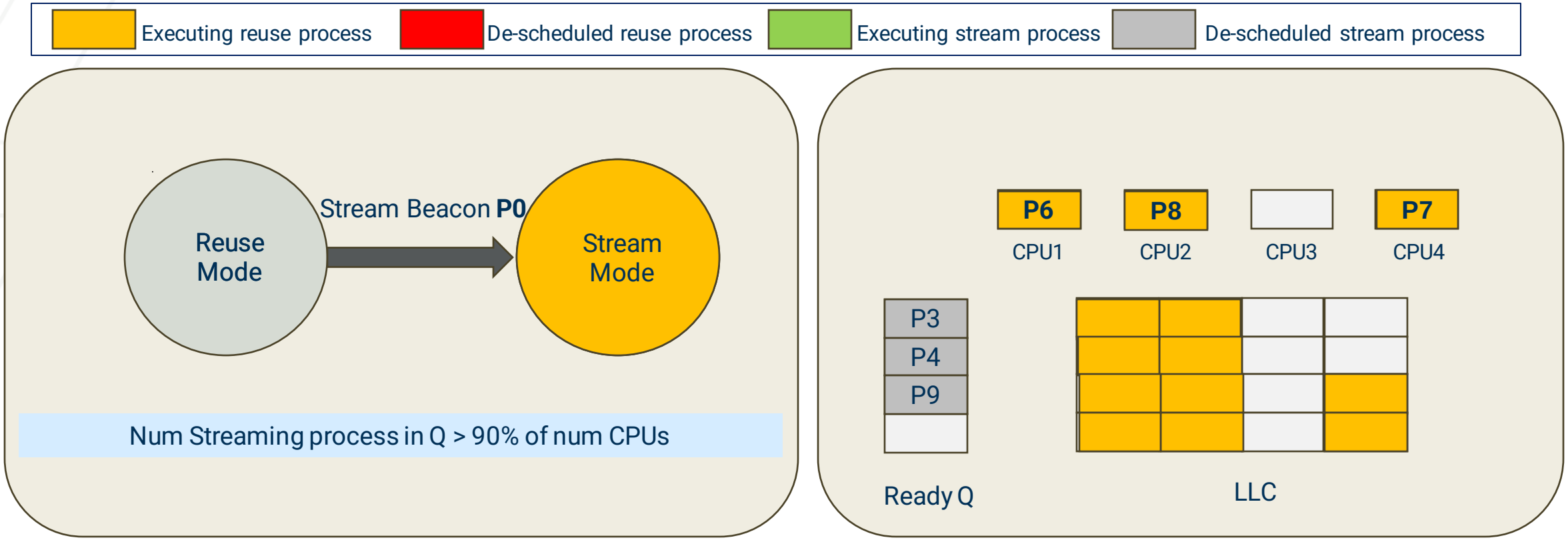
Working of The Beacon Scheduler



Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

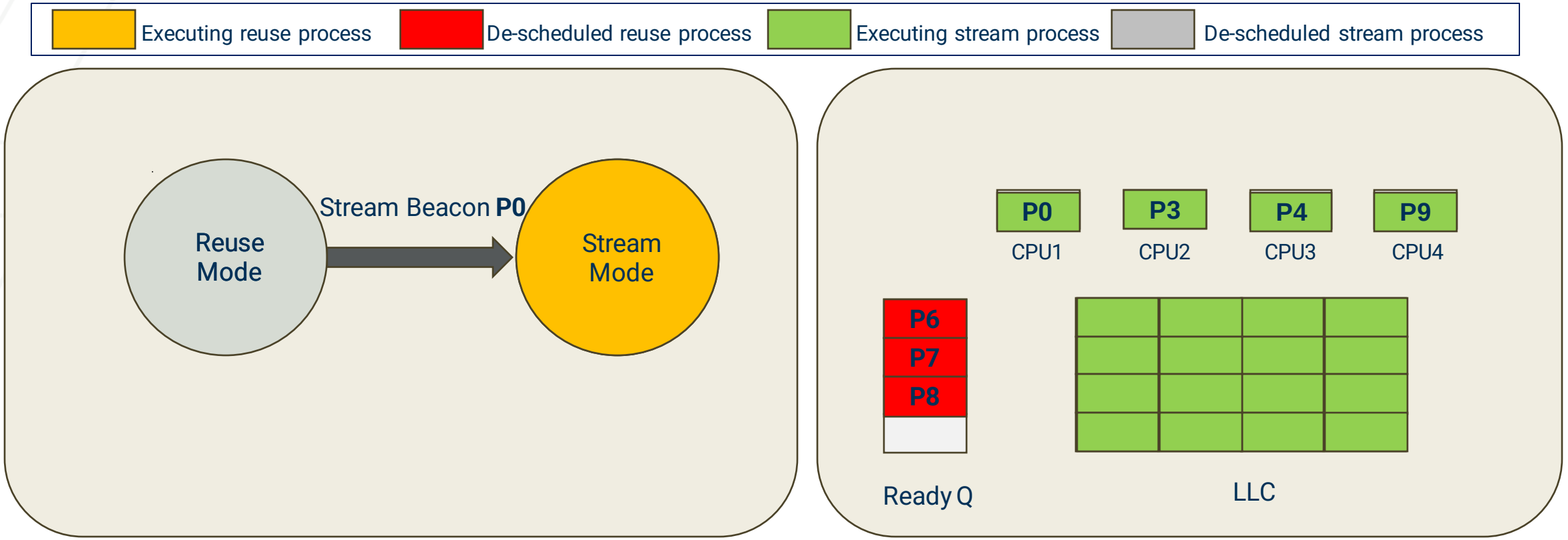
Working of The Beacon Scheduler



Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Working of The Beacon Scheduler

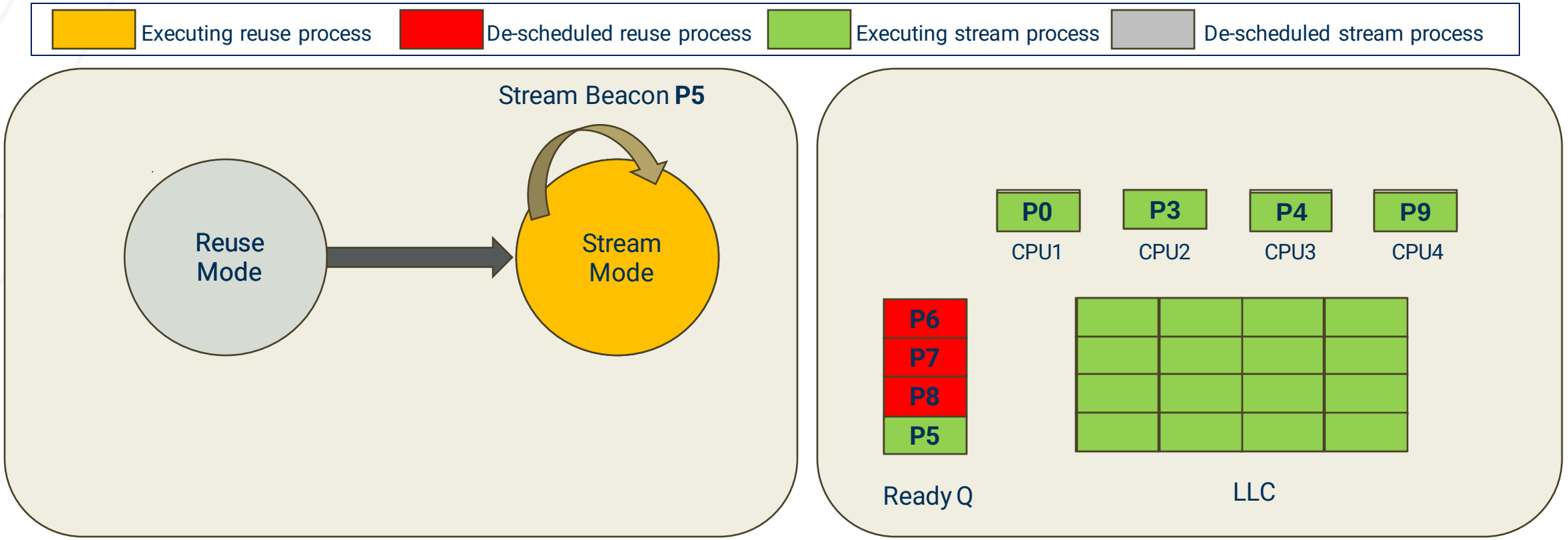


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

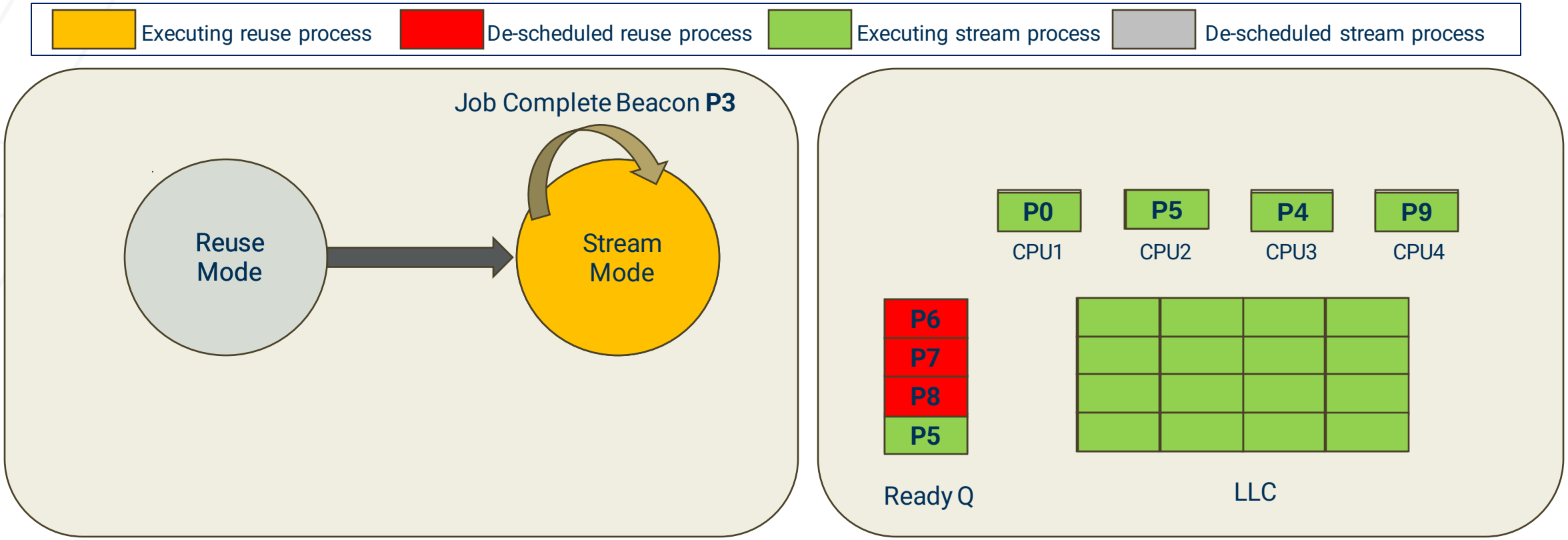


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

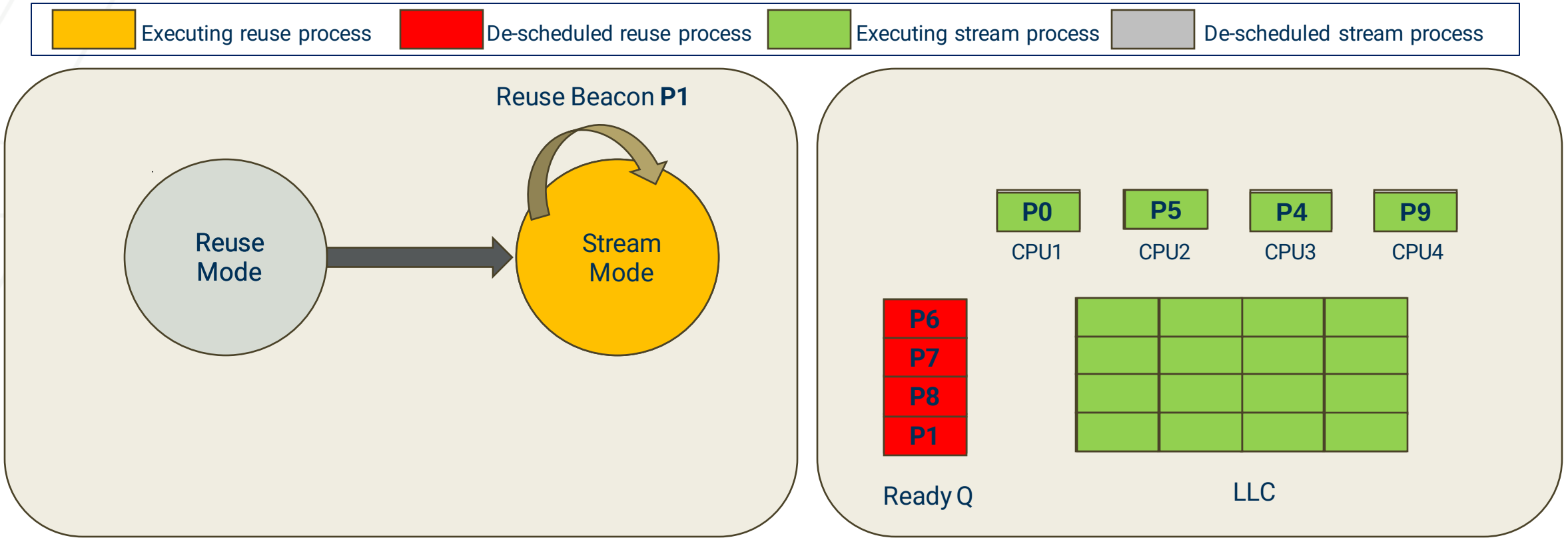


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

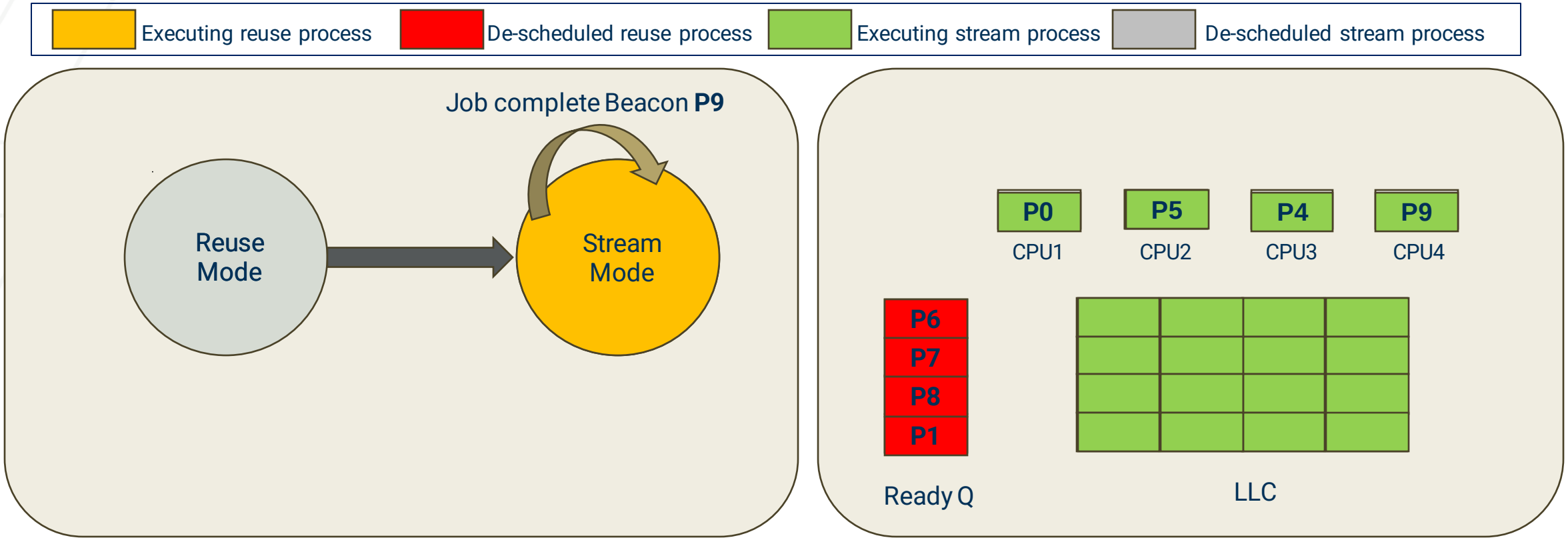


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

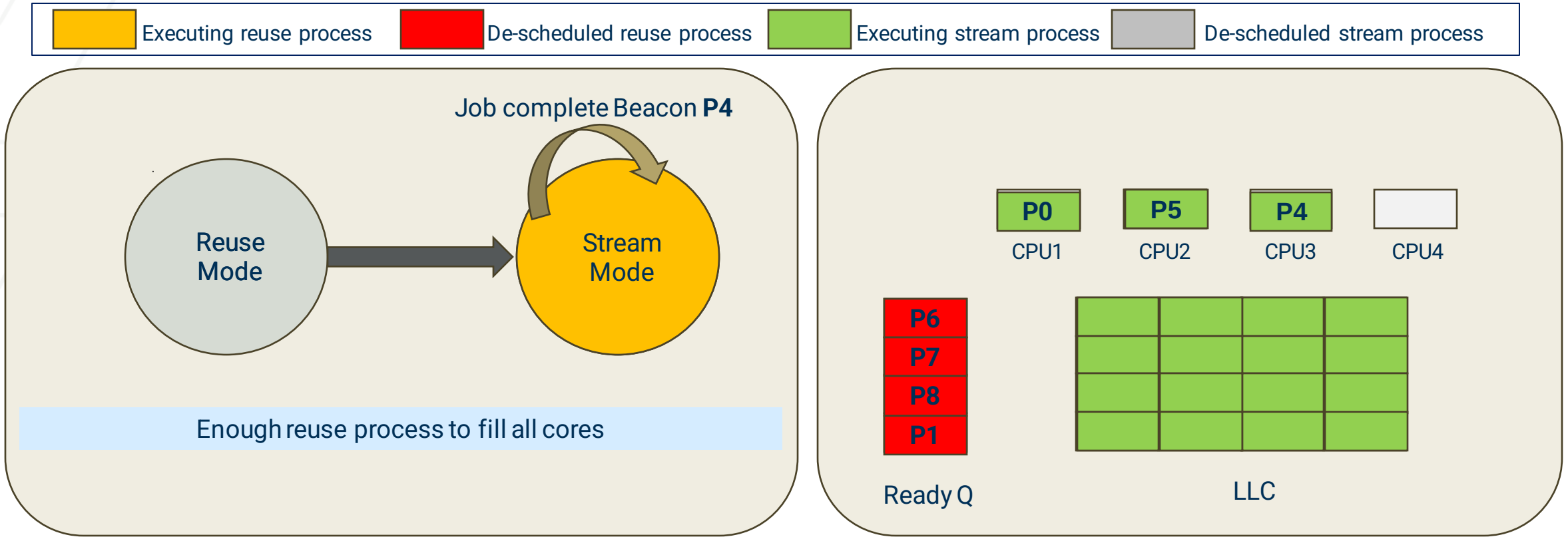


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

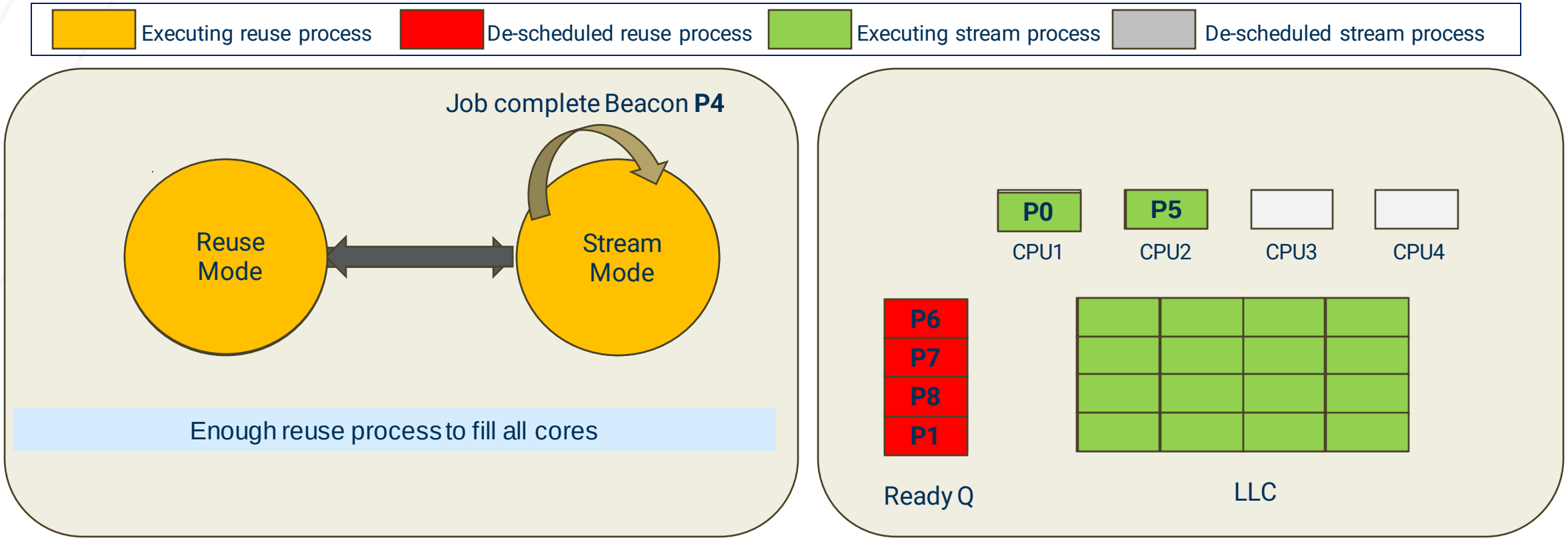


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler

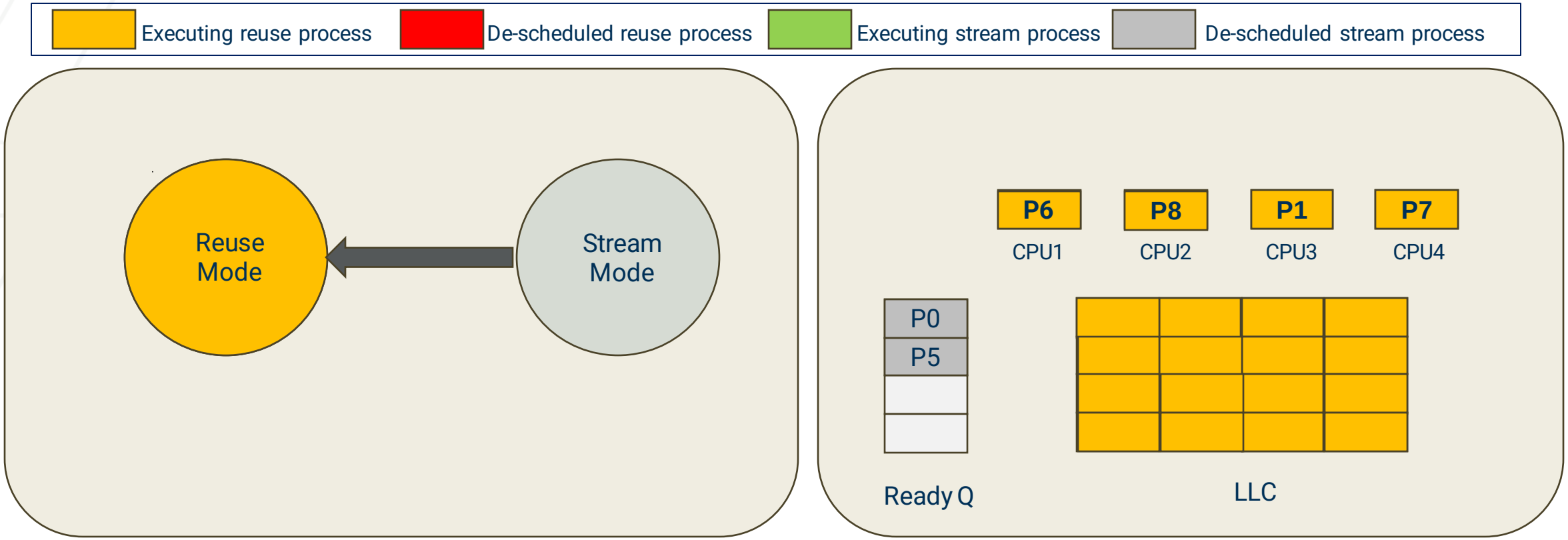


Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

Georgia Tech

Working of The Beacon Scheduler



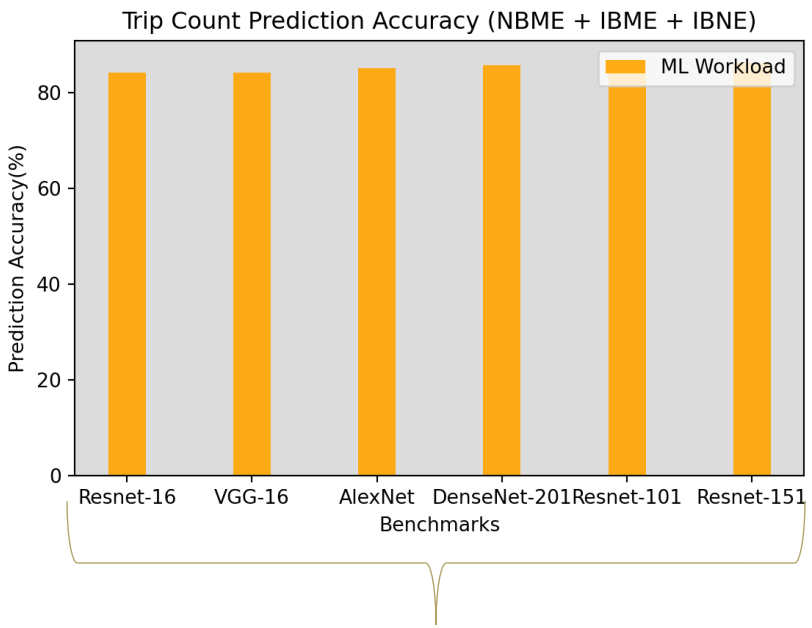
Simplified State Mealy Machine of the BES scheduler

System with Beacons Scheduler

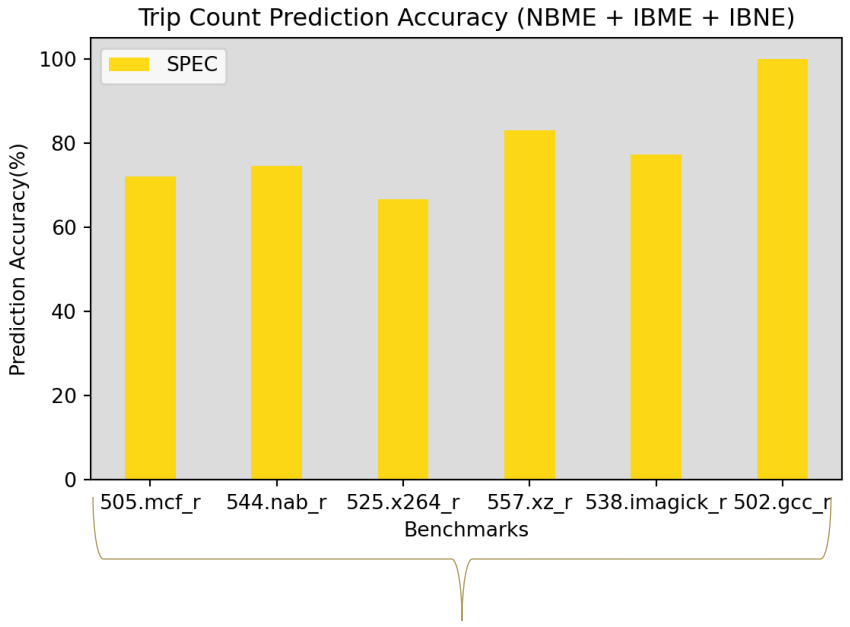
Georgia Tech

Loop Trip Count Prediction: Prediction Accuracy

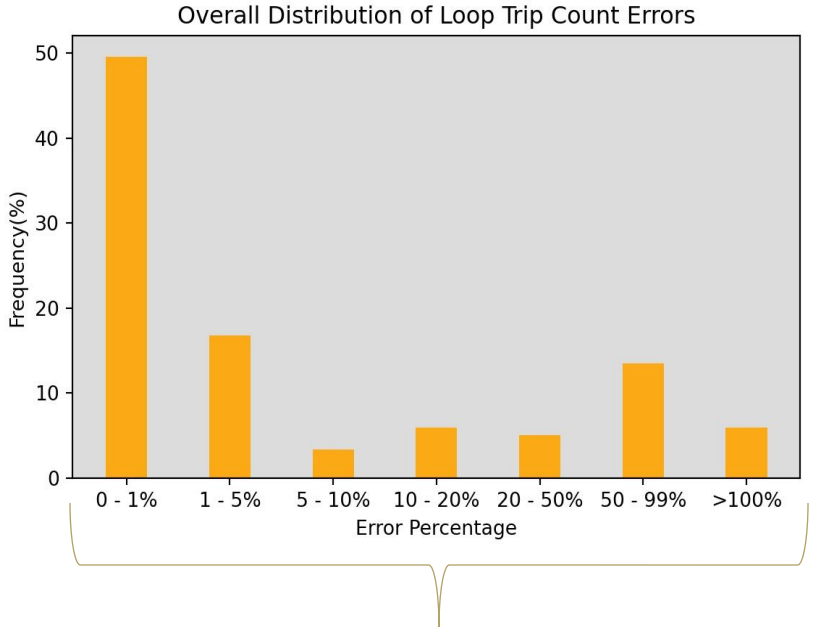
- To predict the loop trip count for irregular loops, *decision trees* classifiers were utilized



High prediction accuracy (>80%) in ML workloads



Moderate prediction accuracy, but with low absolute error



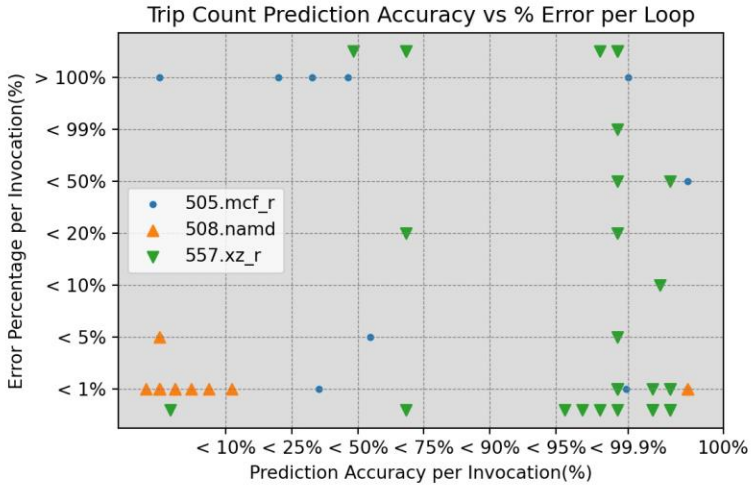
Majority (>70%) of the trip count absolute errors lies is less than 5%

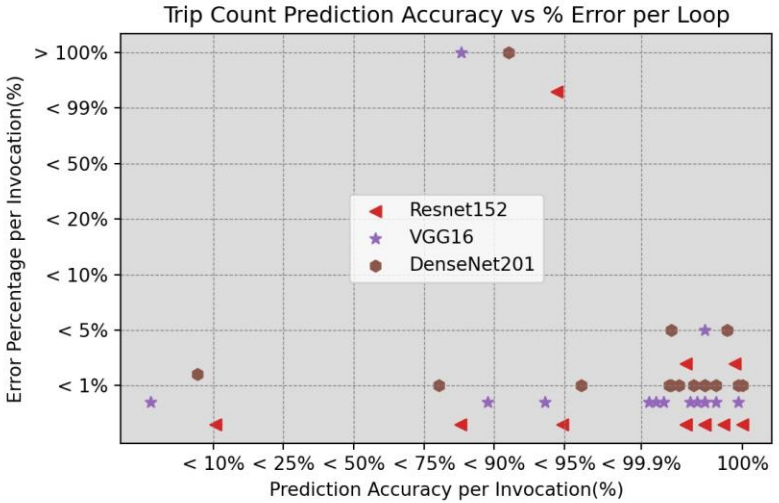
- Most Neural Network-based ML workloads can be broadly dissected into several constituent layers (convolutional layer, fc layer, etc).
- Each of these layers performs an unique set of operations, that can be abstracted throughout different input sets, by *learning the correlation between loop trip counts and its critical variables*.

Georgia Tech

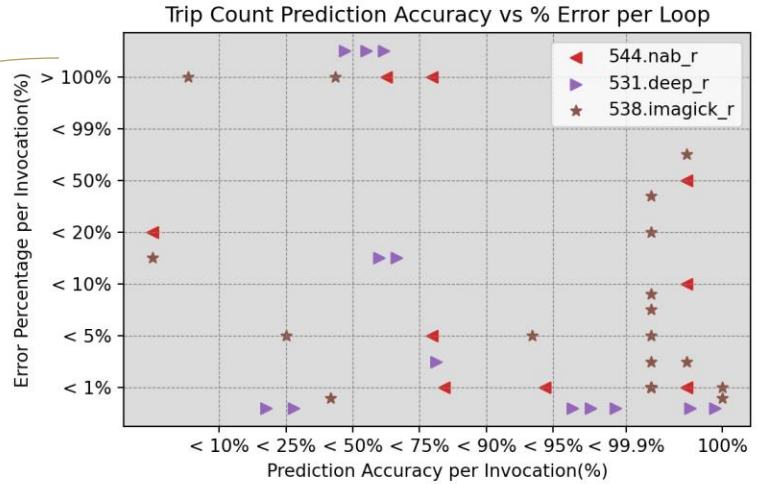
Loop Trip Count Prediction: Absolute Errors

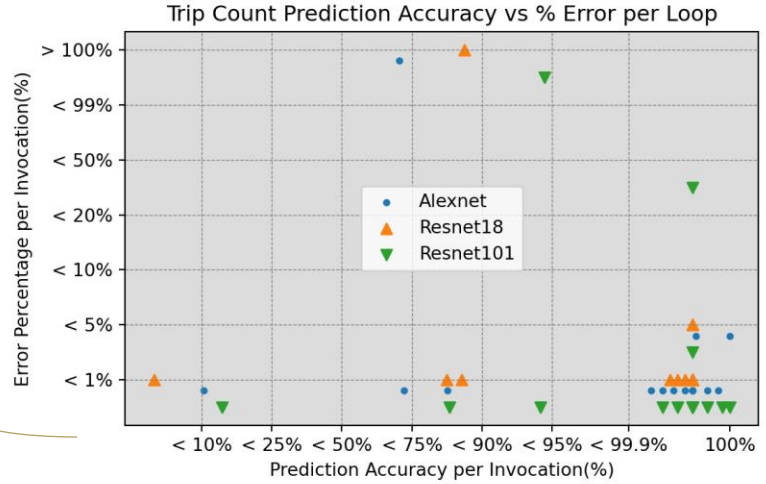
- Absolute error helps us determine the extent of trip count misprediction





Low prediction accuracy co-relates with low absolute errors

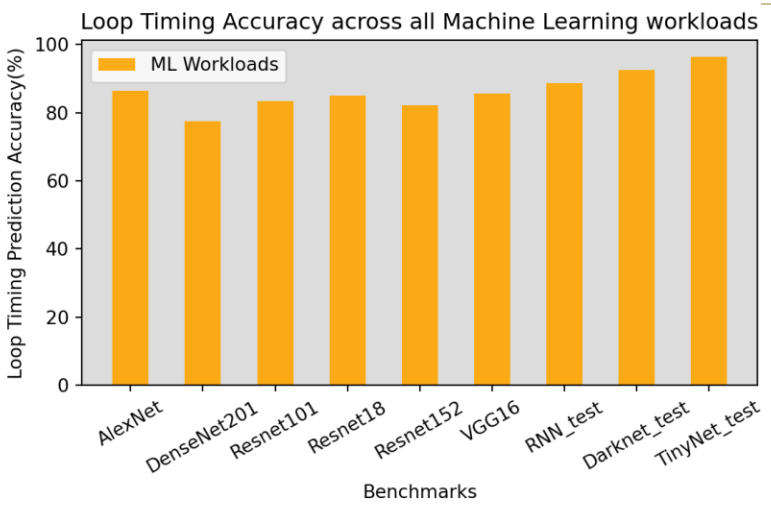


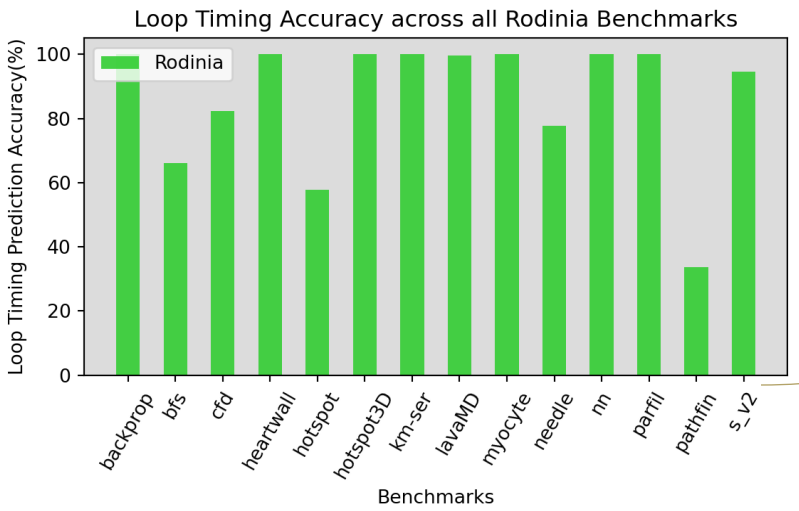


Georgia Tech

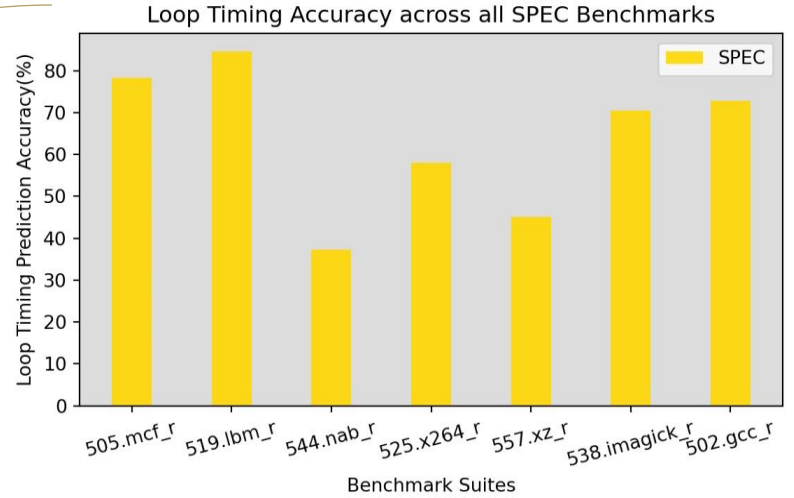
Loop Timing Prediction

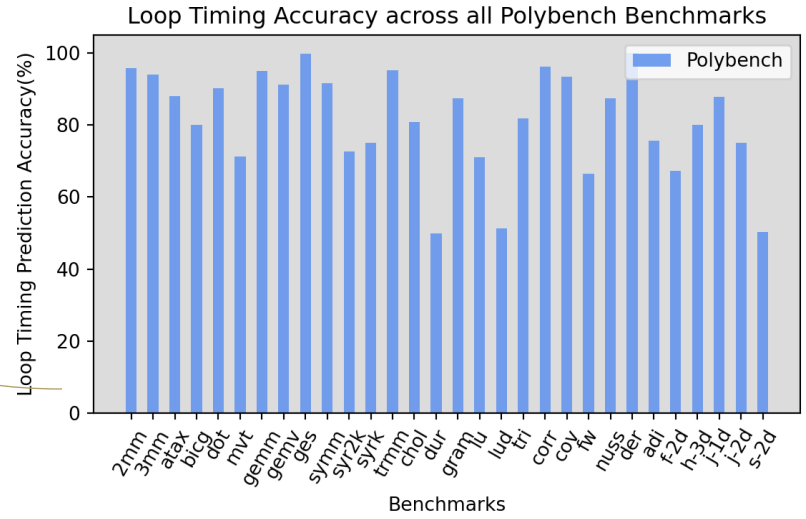
- Loop timing helps us determine the extent of overlap between different loop phases





Correlation between trip count accuracy & timing accuracy is observed

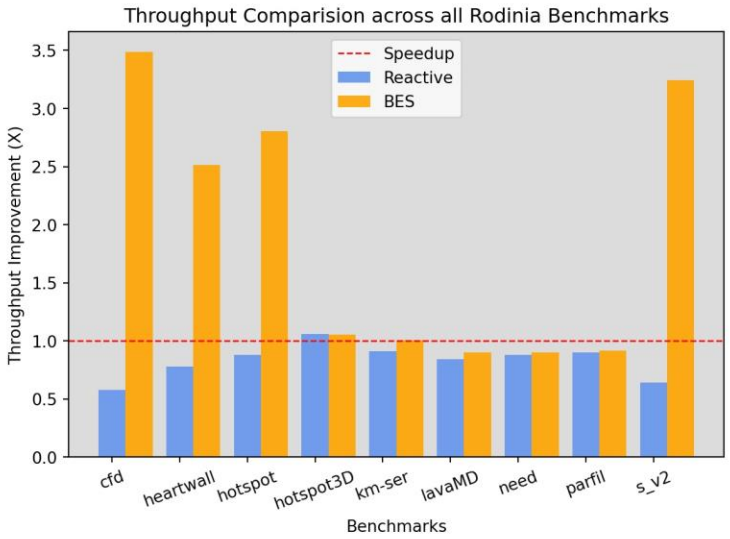


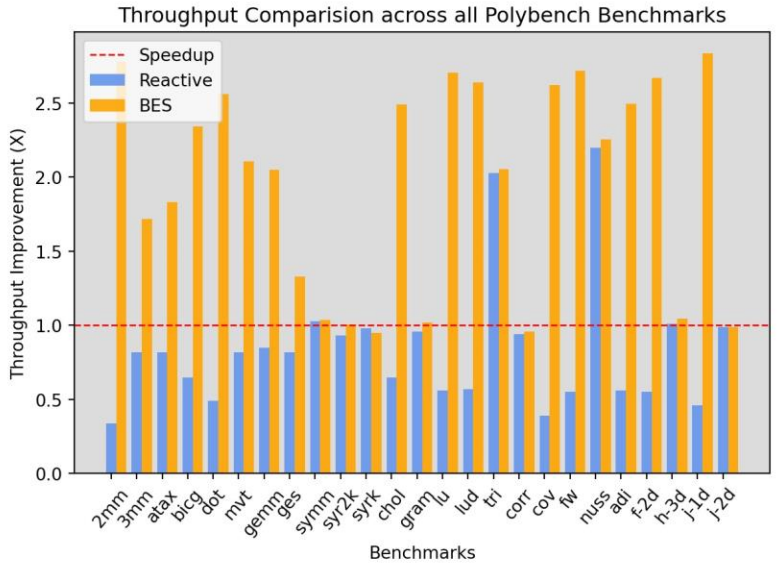


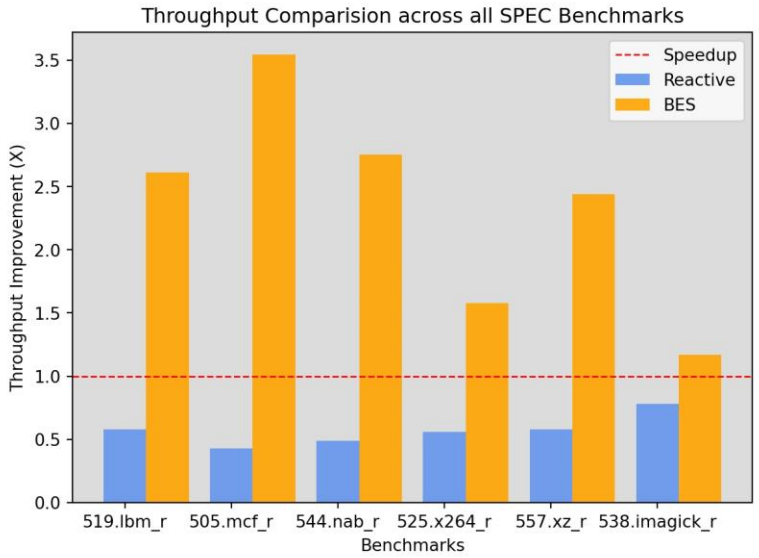
Georgia Tech

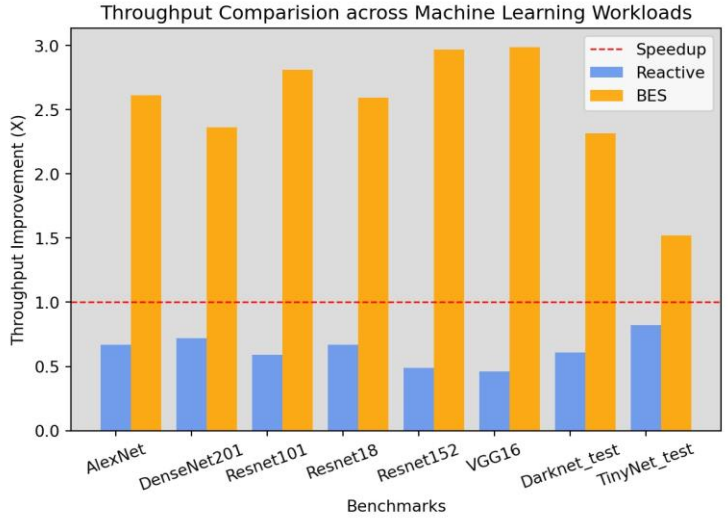
Proactive Workload Scheduling

- Average Throughput Improvement of 2.62x over Merlin Scheduler, and 1.92x over CFS









Georgia Tech

Conclusion

- **Beacons Framework**, an end-to-end system that estimates dynamic loop characteristics and leverages them to solve the workload scheduling problem efficiently

- Devised novel analysis to estimate the following loop characteristics with over 80% prediction accuracy:
    - *Loop Trip Count*
    - *Loop Timing*
    - *Loop Memory Footprint*

- Improved the throughput over CFS by 1.92x and over Merlin Scheduler by 2.62x

Georgia Tech