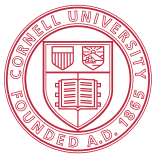
Dynamic Hazard Resolution for Pipelining Irregular Loops in High-Level Synthesis

Steve Dai[1], Ritchie Zhao[1], Gai Liu[1], Shreesha Srinath[1], Udit Gupta[2], Christopher Batten[1], Zhiru Zhang[1]

[1] Electrical and Computer Engineering, Cornell University
[2] Computer Science, Harvard University



Cornell University

CSL

Loop Pipelining in High-Level Synthesis

- Ultimate Goal
  - Synthesize an efficient pipeline with the highest possible throughput for all kinds of applications
- Conventional HLS pipelining
  - Ineffective for loops with data-dependent memory accesses
    - Cannot be predicted by static compiler analysis
    - Infrequent dynamic structural and data hazards
- Our Study
  - Augment HLS synthesized pipeline with application-specific dynamic hazard resolution logic

Loop Pipelining in High-Level Synthesis

- Ultimate Goal
  - Synthesize an efficient pipeline with the highest possible throughput for all kinds of applications
- Conventional HLS pipelining
  - Ineffective for loops with data-dependent memory accesses
    - Cannot be predicted by static compiler analysis
    - Infrequent dynamic structural and data hazards
- Our Study
  - Augment HLS synthesized pipeline with application-specific dynamic hazard resolution logic

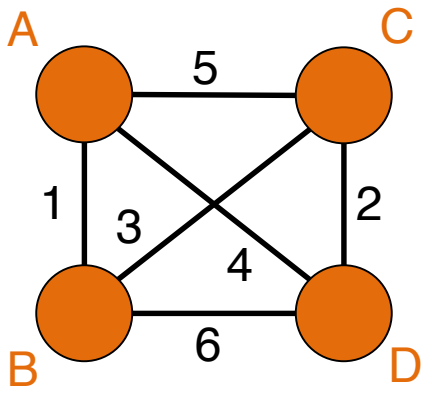
Loop Pipelining in High-Level Synthesis

- Ultimate Goal
  - Synthesize an efficient pipeline with the highest possible throughput for all kinds of applications
- Conventional HLS pipelining
  - Ineffective for loops with data-dependent memory accesses
    - Cannot be predicted by static compiler analysis
    - Infrequent dynamic structural and data hazards
- Our Study
  - Augment HLS synthesized pipeline with application-specific dynamic hazard resolution logic

Pipelining Loops with Infrequent Hazards

Maximal Matching

```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```



j=1

j=2

j=3

j=4

j=5

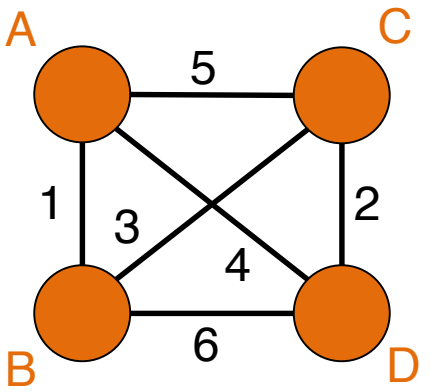
j=6

Clock Cycles

Pipelining Loops with Infrequent Hazards

Maximal Matching

```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```



j=1

j=2

j=3

j=4

j=5

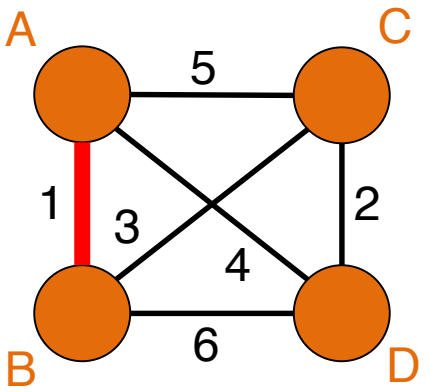
j=6

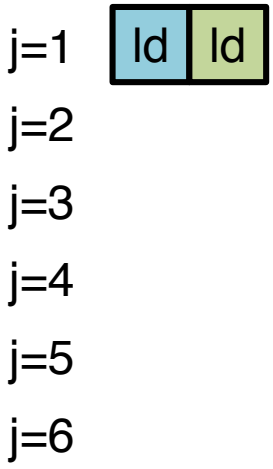
Clock Cycles

Pipelining Loops with Infrequent Hazards

Maximal Matching

```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

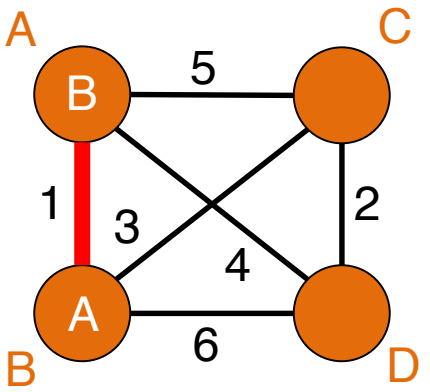




Pipelining Loops with Infrequent Hazards

Maximal Matching

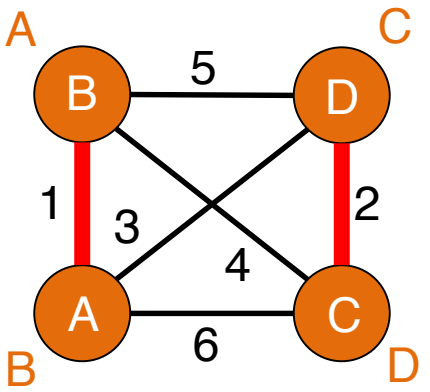
```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

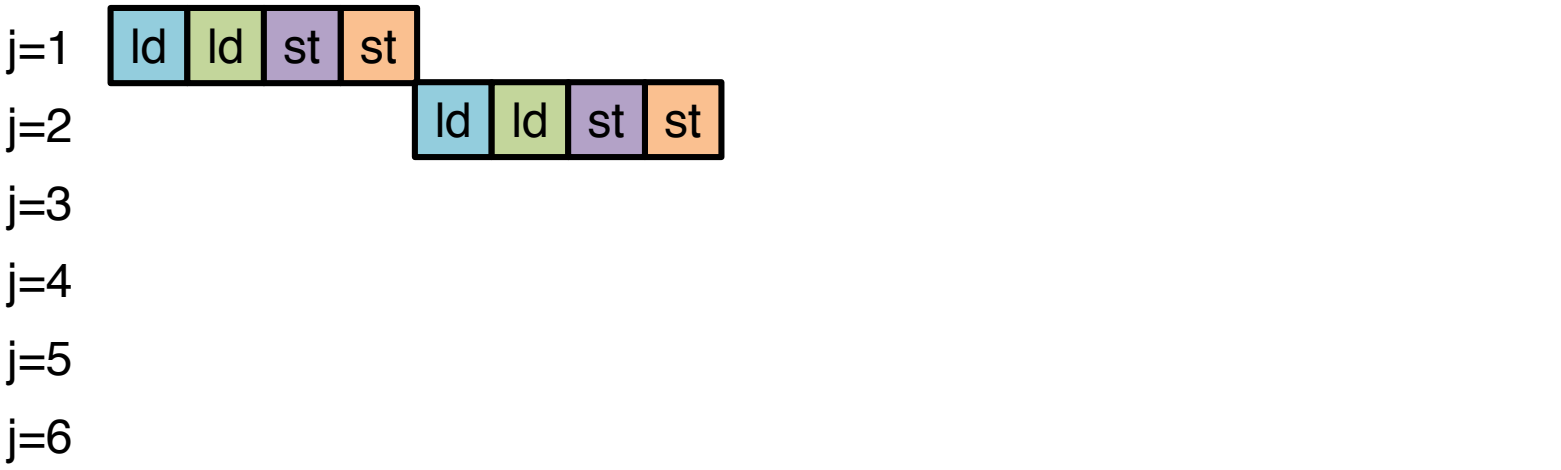


Pipelining Loops with Infrequent Hazards

Maximal Matching

```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

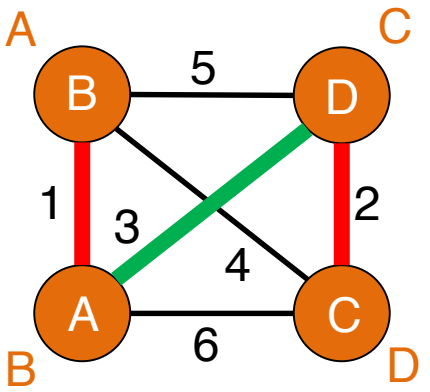


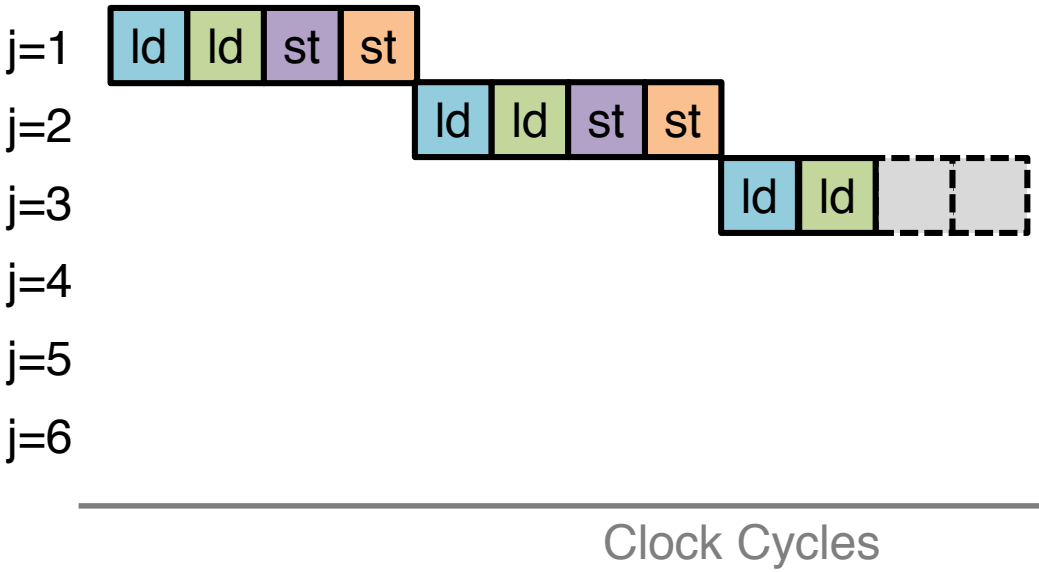


Pipelining Loops with Infrequent Hazards

Maximal Matching

```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

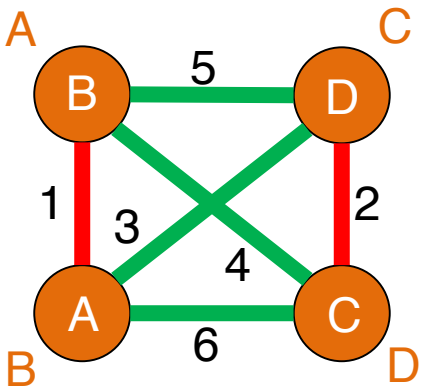


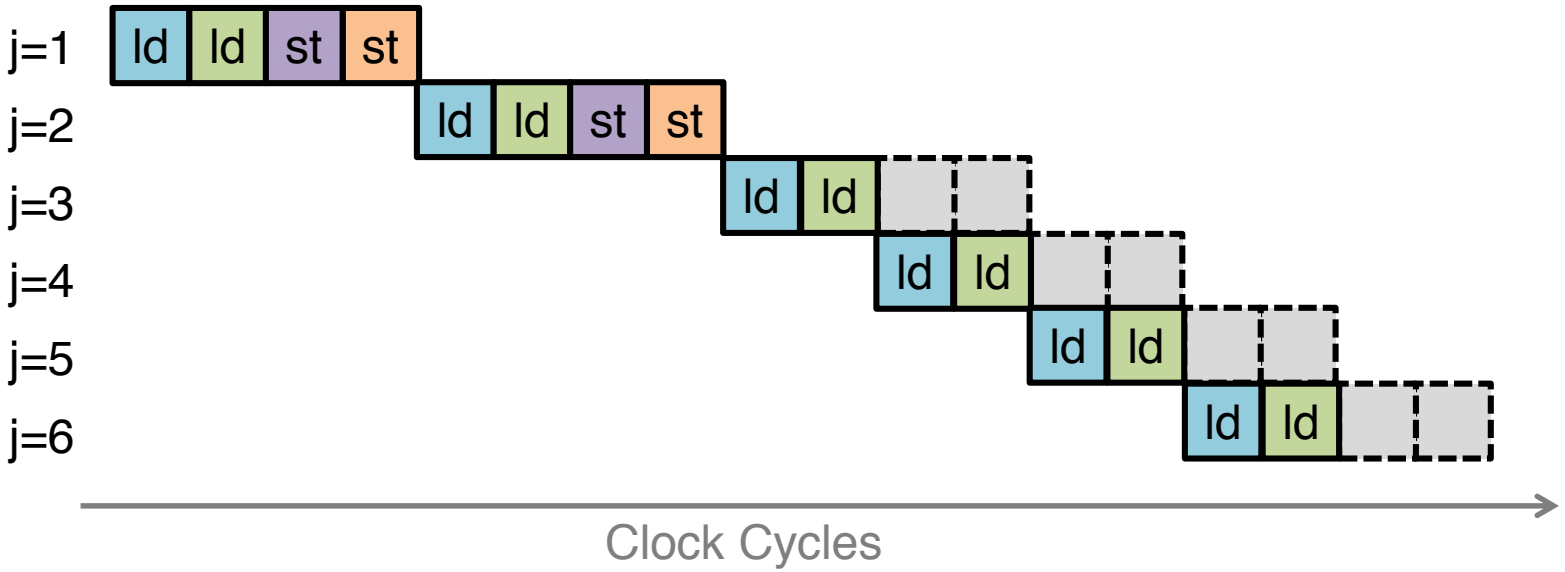


Pipelining Loops with Infrequent Hazards

Maximal Matching

```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

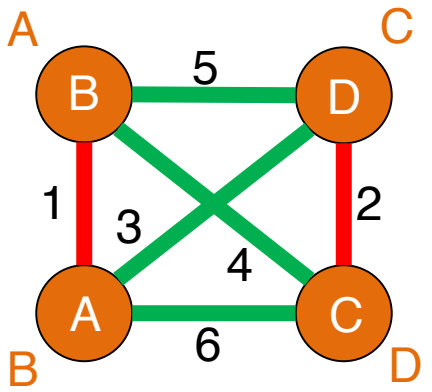




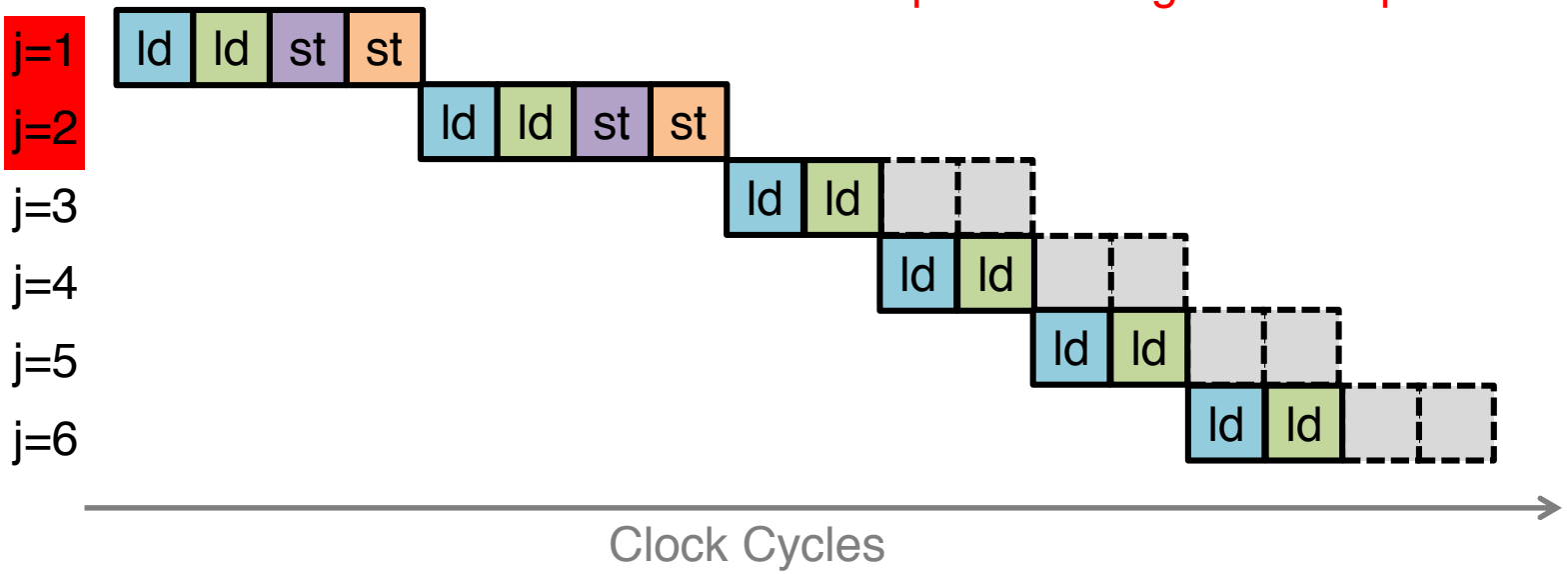
Pipelining Loops with Infrequent Hazards

Maximal Matching

```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```



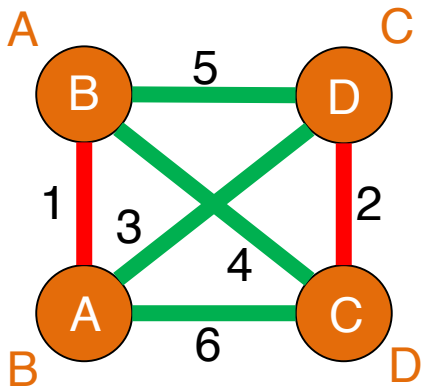
Independent edges → Requires stores



Pipelining Loops with Infrequent Hazards

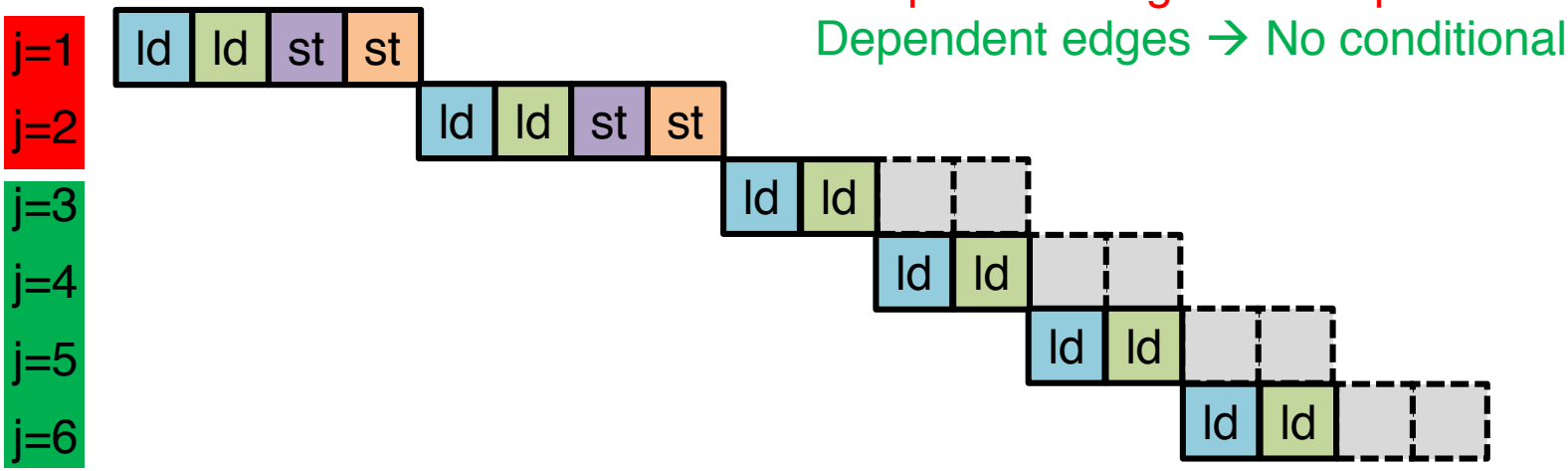
Maximal Matching

```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```



Independent edges → Requires stores

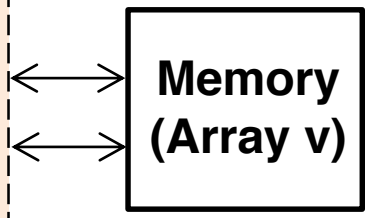
Dependent edges → No conditional stores

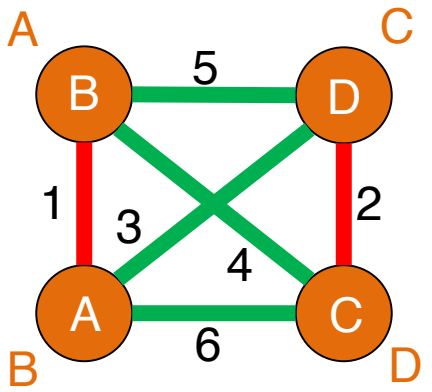


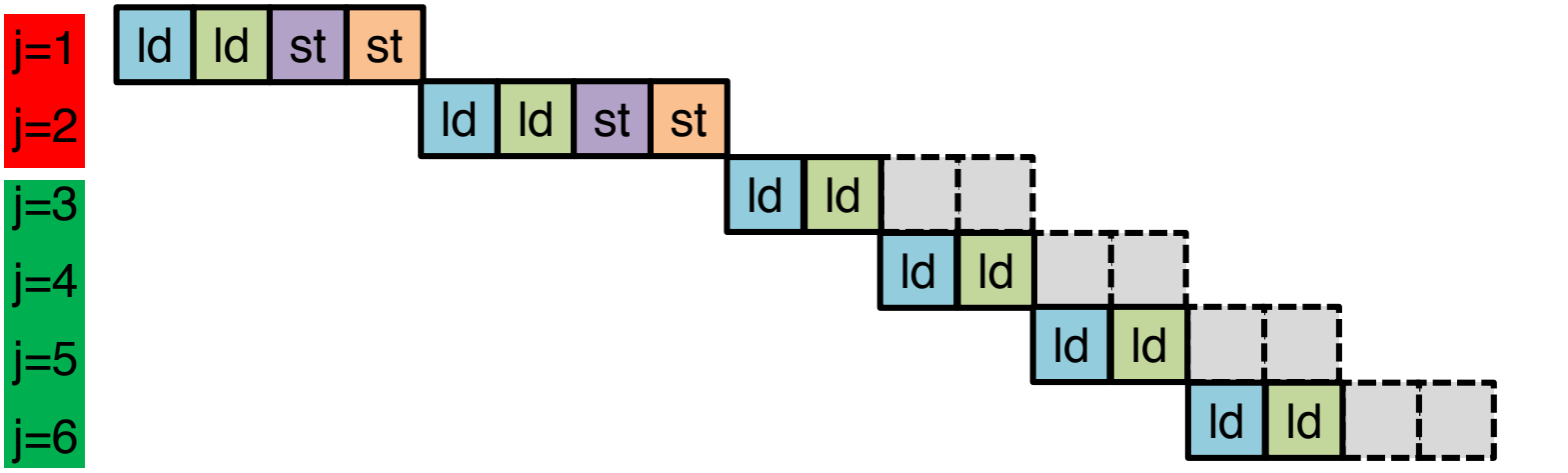
Pipelining Loops with Infrequent Hazards

Maximal Matching

```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```





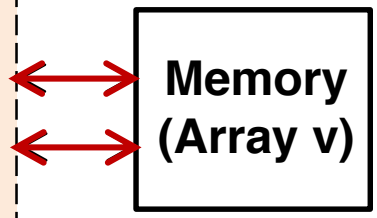


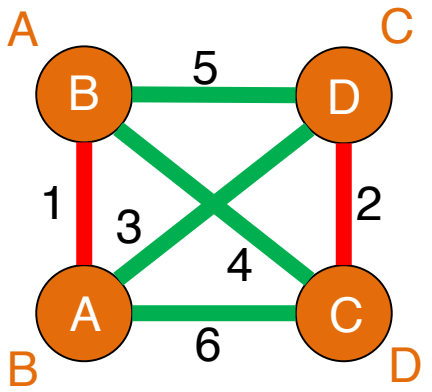
Pipelining Loops with Infrequent Hazards

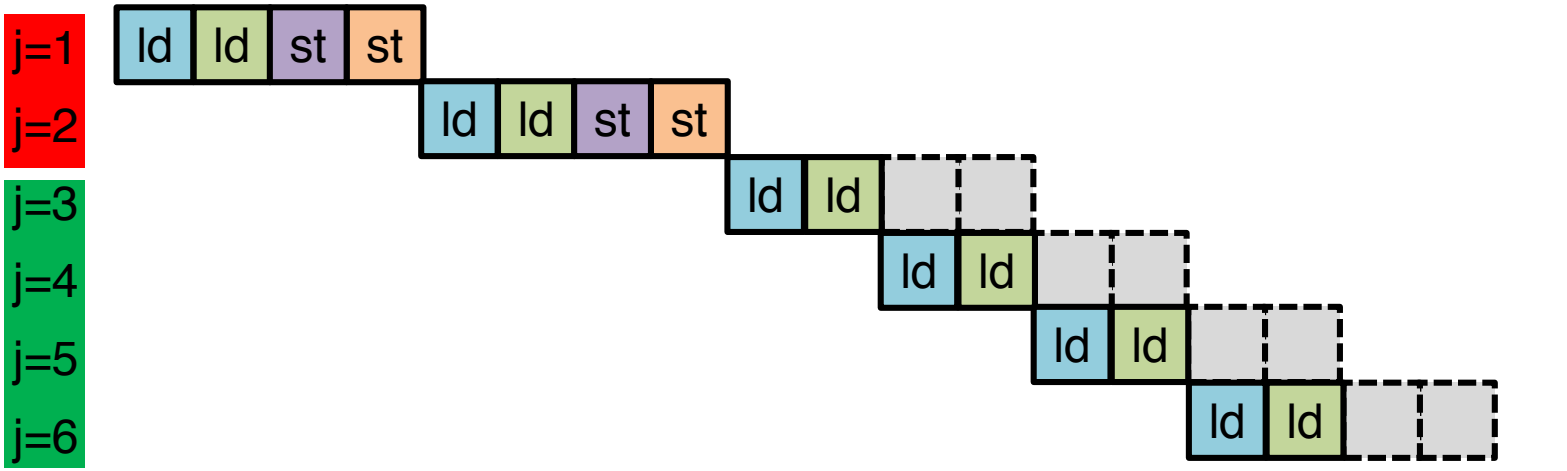
Maximal Matching

```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if (!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

structural hazards (memory ports)







Pipelining Loops with Infrequent Hazards

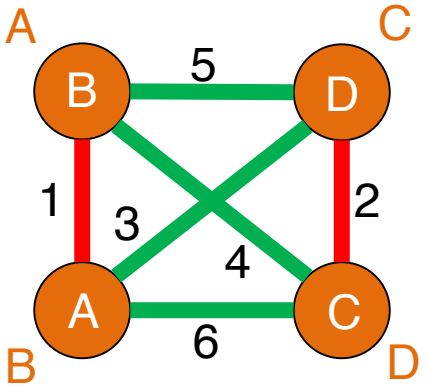
Maximal Matching

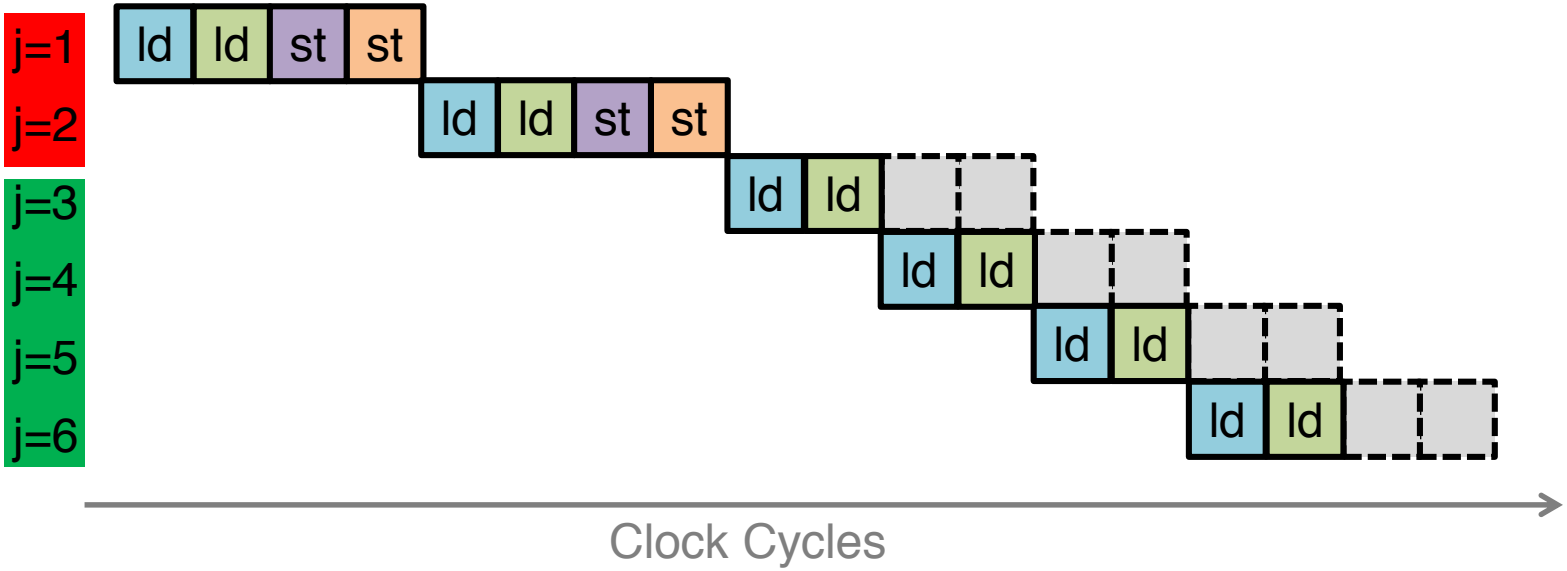
```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if(!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

Memory (Array v)

Data hazards (read-after-write)

structural hazards (memory ports)





Pipelining Loops with Infrequent Hazards

Maximal Matching

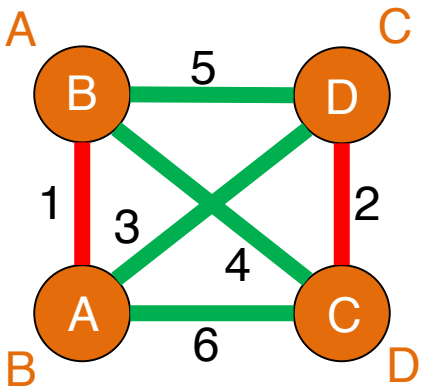
```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if(!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

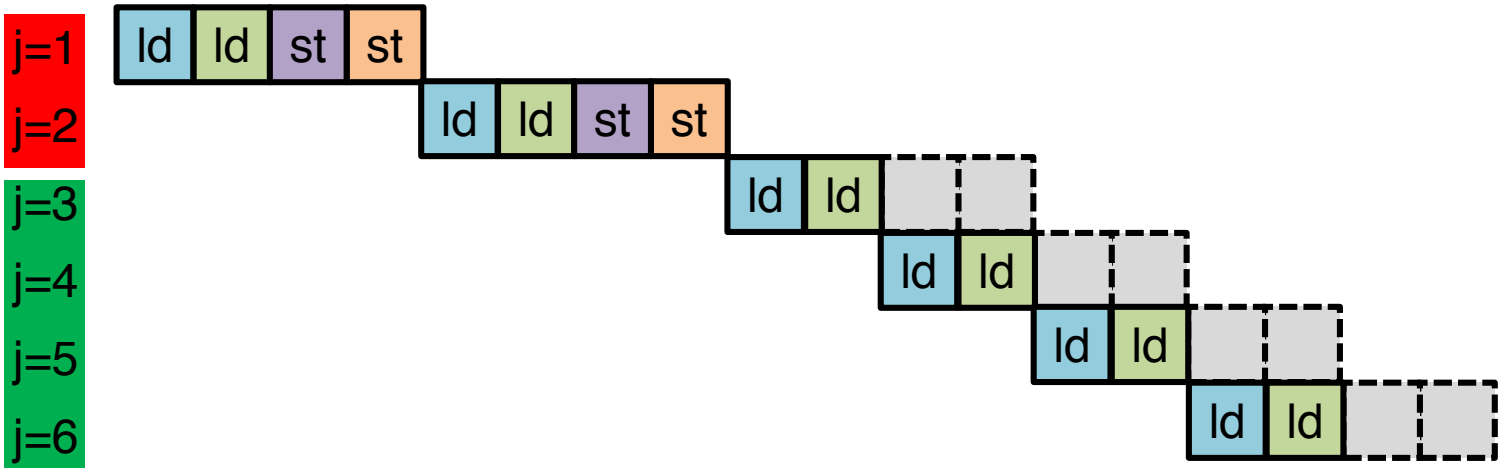
Memory (Array v)

Data hazards (read-after-write)

structural hazards (memory ports)

Occur infrequently





Pipelining Loops with Infrequent Hazards

Maximal Matching

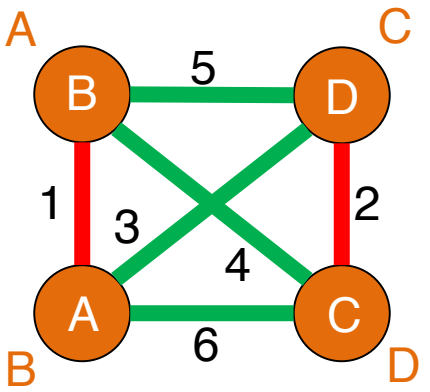
```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if(!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

Memory (Array v)

Data hazards (read-after-write)

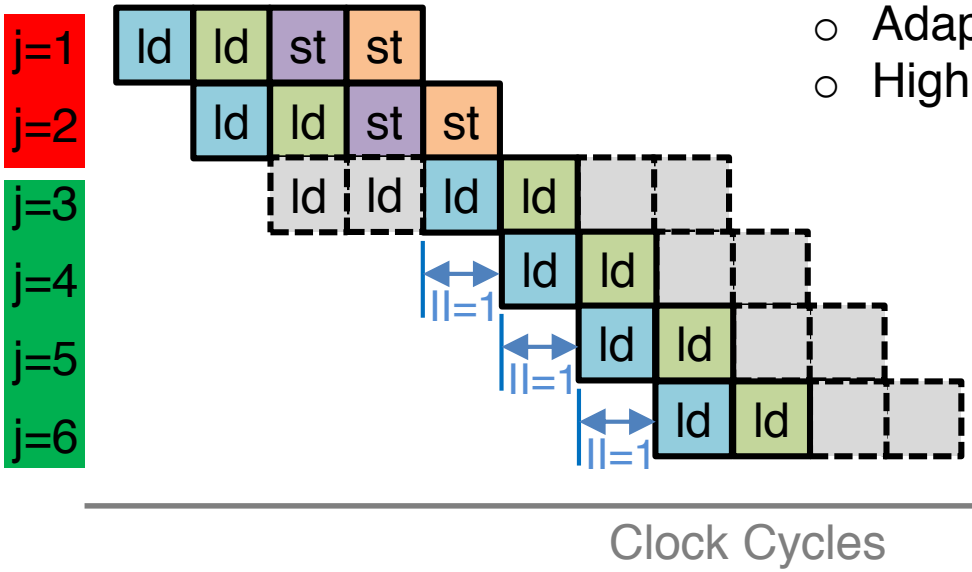
structural hazards (memory ports)

Occur infrequently



Optimal Schedule

- Adapts to dynamic infrequent hazards
- High throughput for most iterations



Pipelining Loops with Infrequent Hazards

Maximal Matching

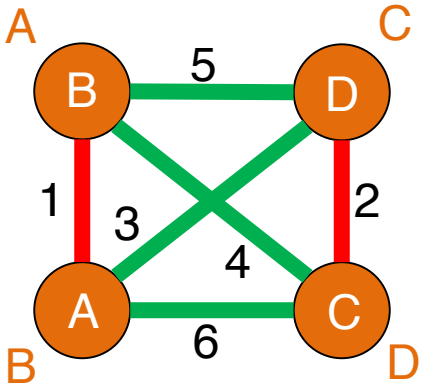
```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if(!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

Memory (Array v)

Data hazards (read-after-write)

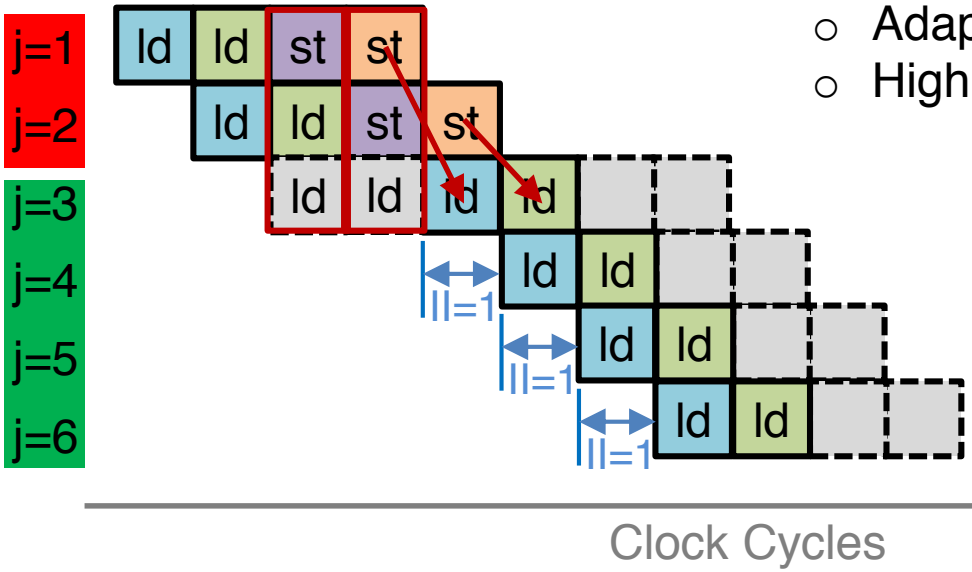
structural hazards (memory ports)

Occur infrequently



Optimal Schedule

- Adapts to dynamic infrequent hazards
- High throughput for most iterations



Pipelining Loops with Infrequent Hazards

Maximal Matching

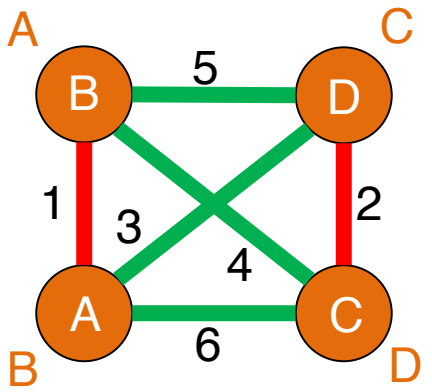
```
for (j=1; j<=6; j++){
  int s = e[j].src;
  int d = e[j].dst;

  if(!v[s] && !v[d]){
    v[s] = d;
    v[d] = s;
}}
```

Memory (Array v)

Data hazards (read-after-write)

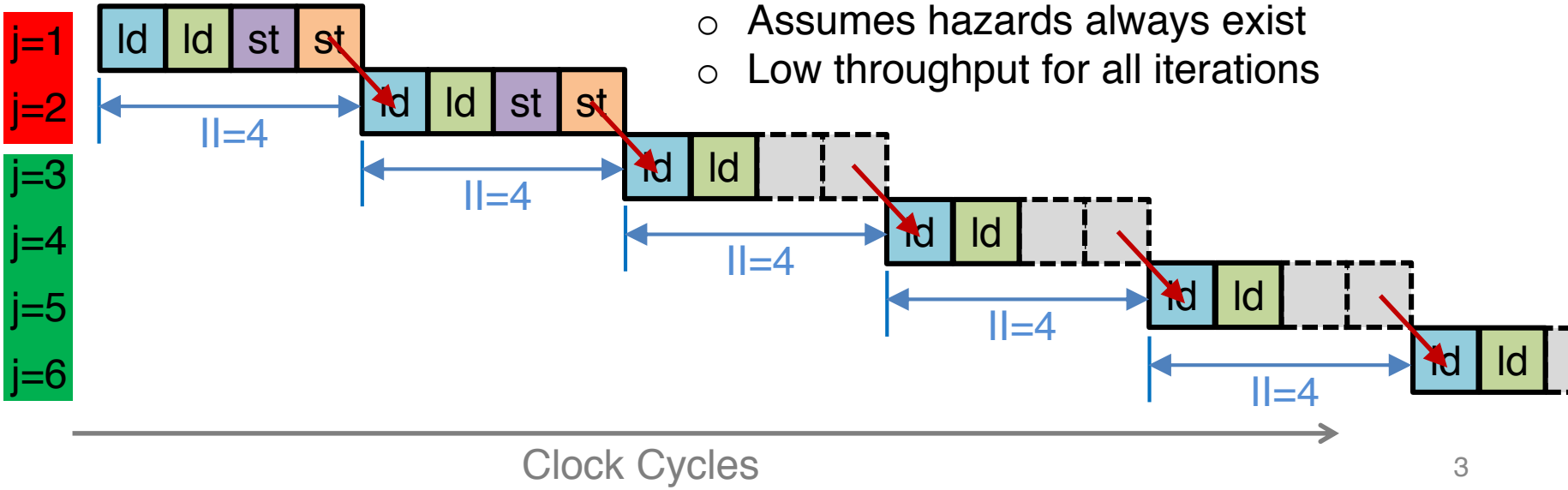
structural hazards (memory ports)

Occur infrequently



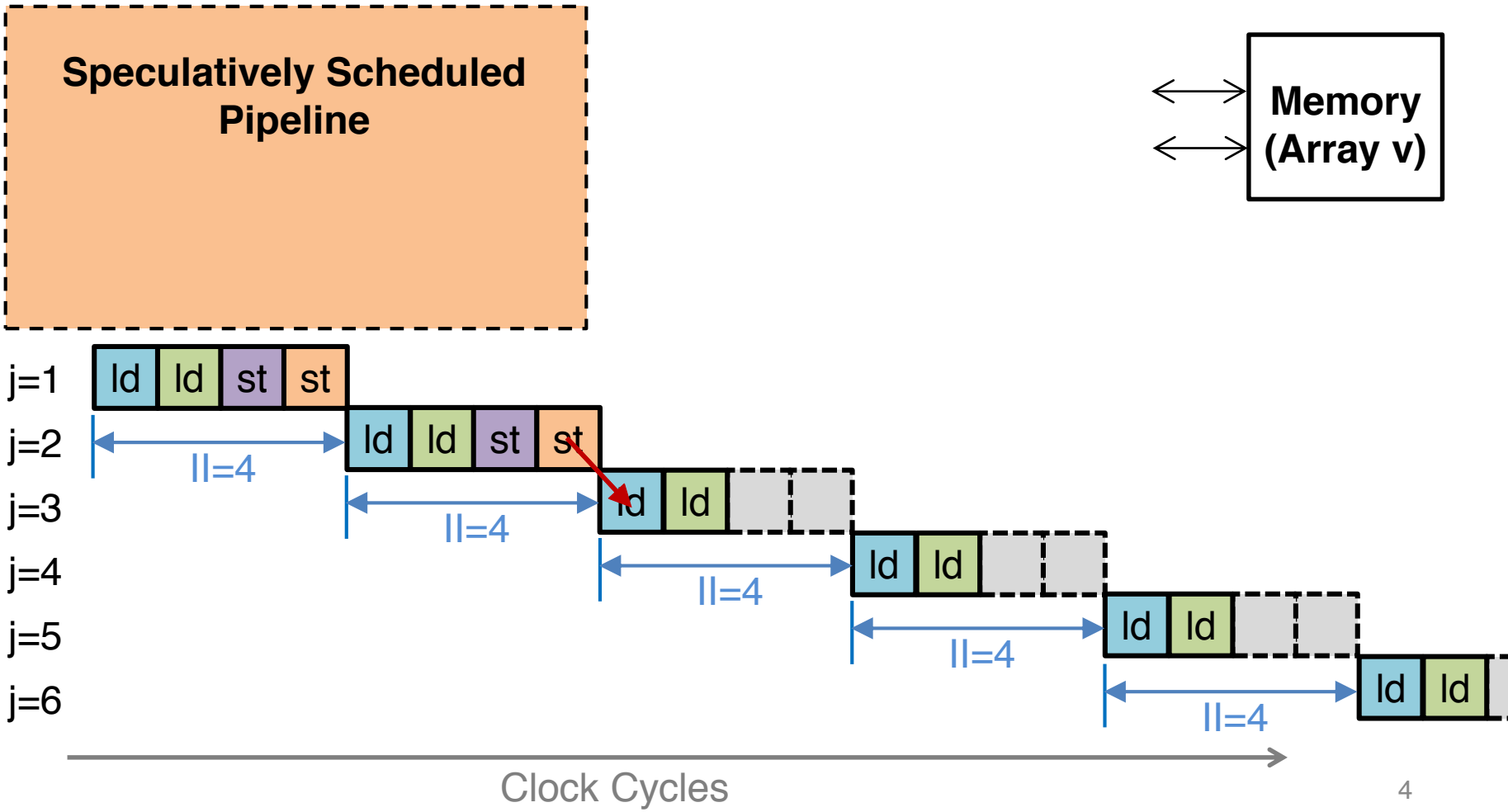
Conventional HLS Schedule

- Assumes hazards always exist
- Low throughput for all iterations



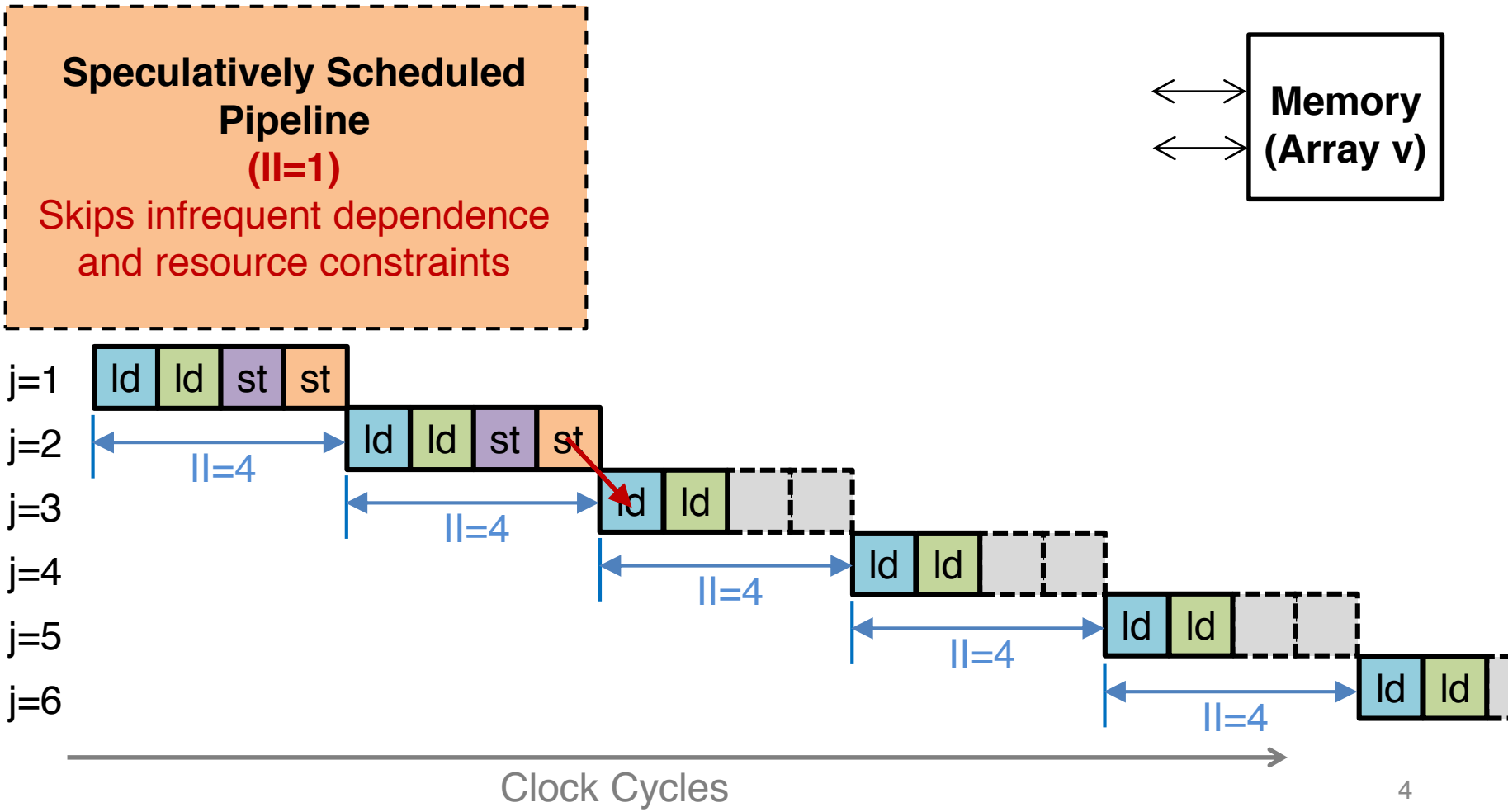
Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime



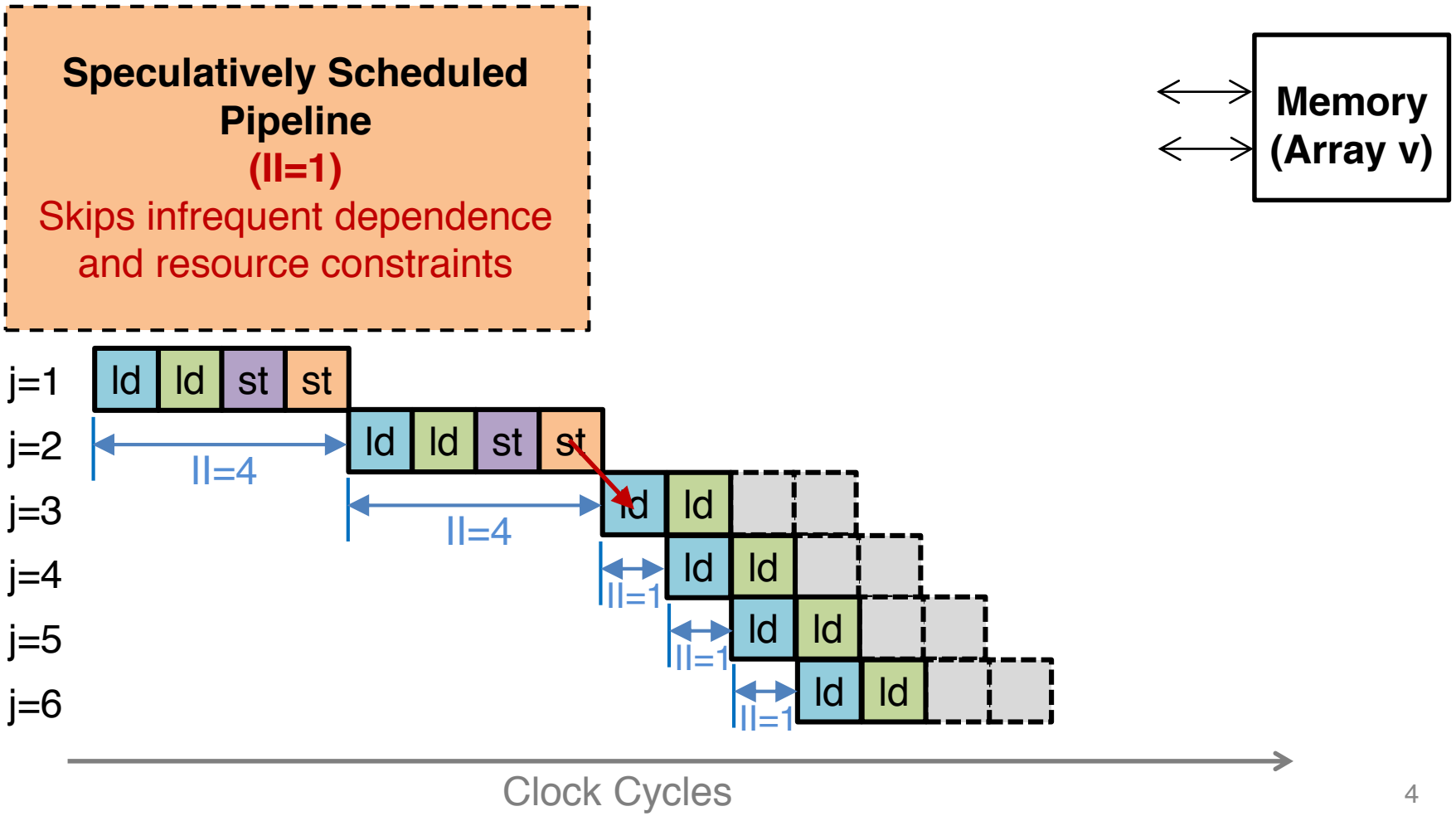
Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime



Dynamic Hazard Resolution for HLS

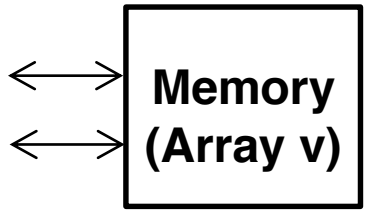
- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime

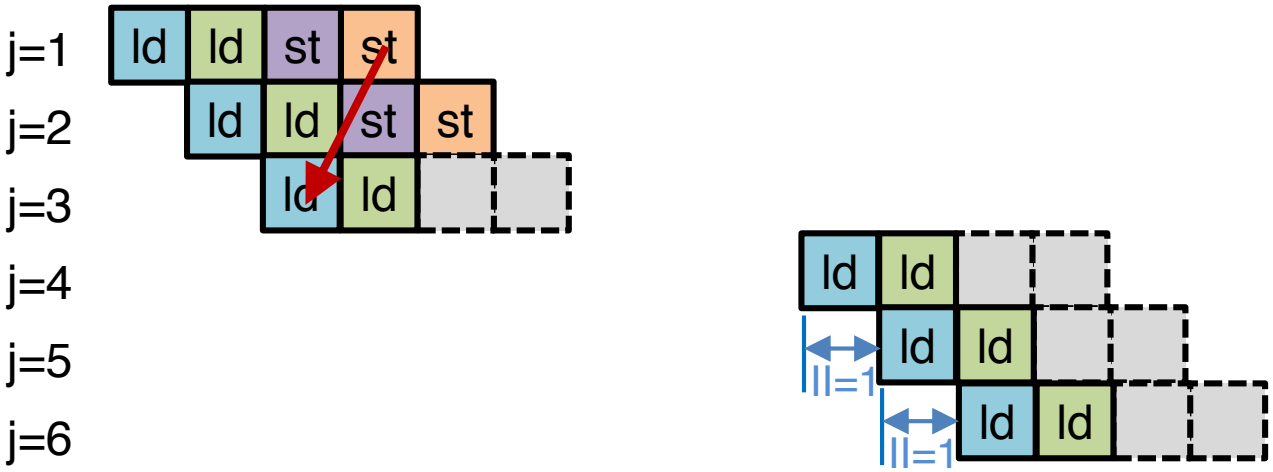


Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime

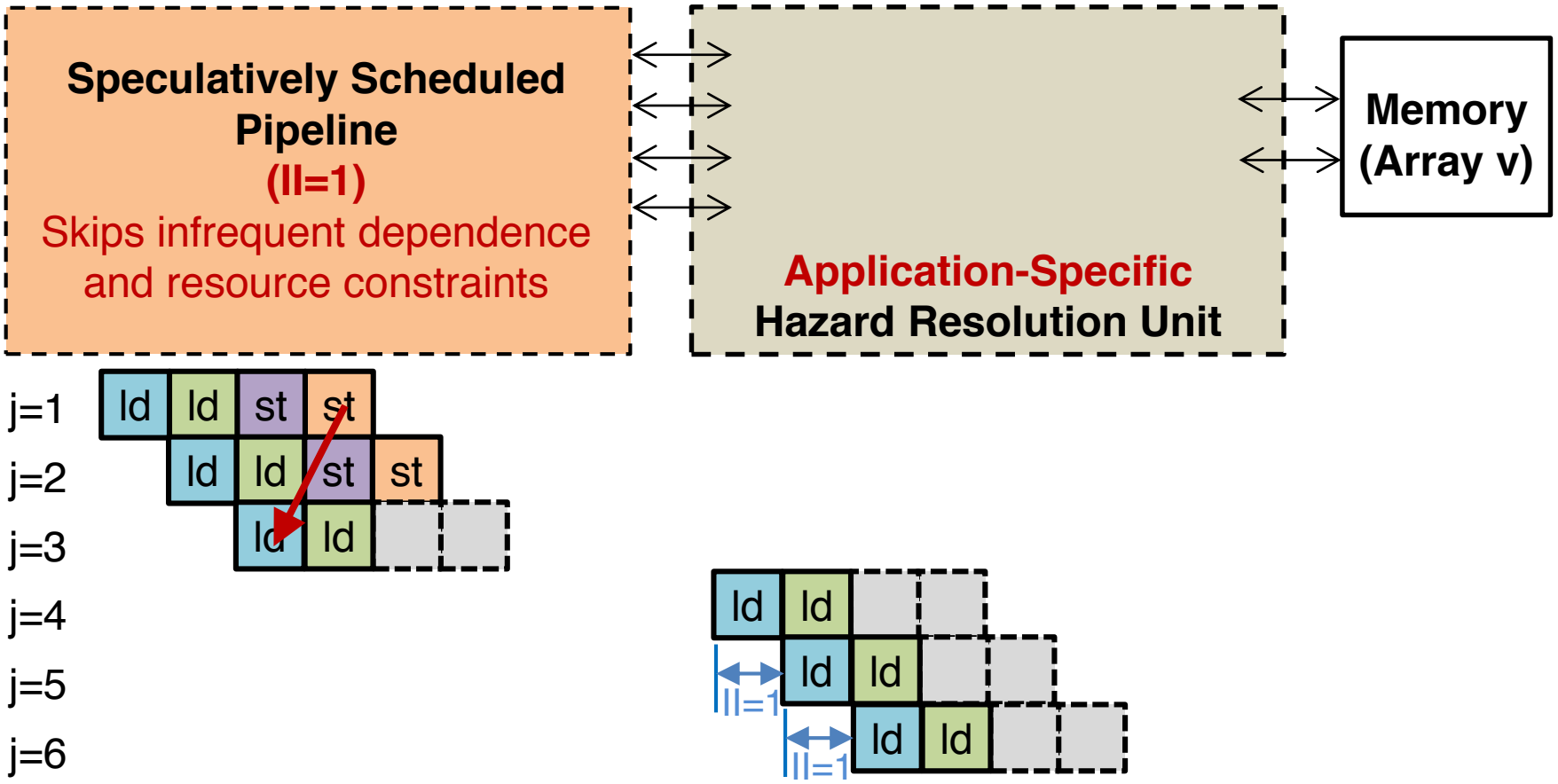
Speculatively Scheduled Pipeline
(II=1)
Skips infrequent dependence and resource constraints





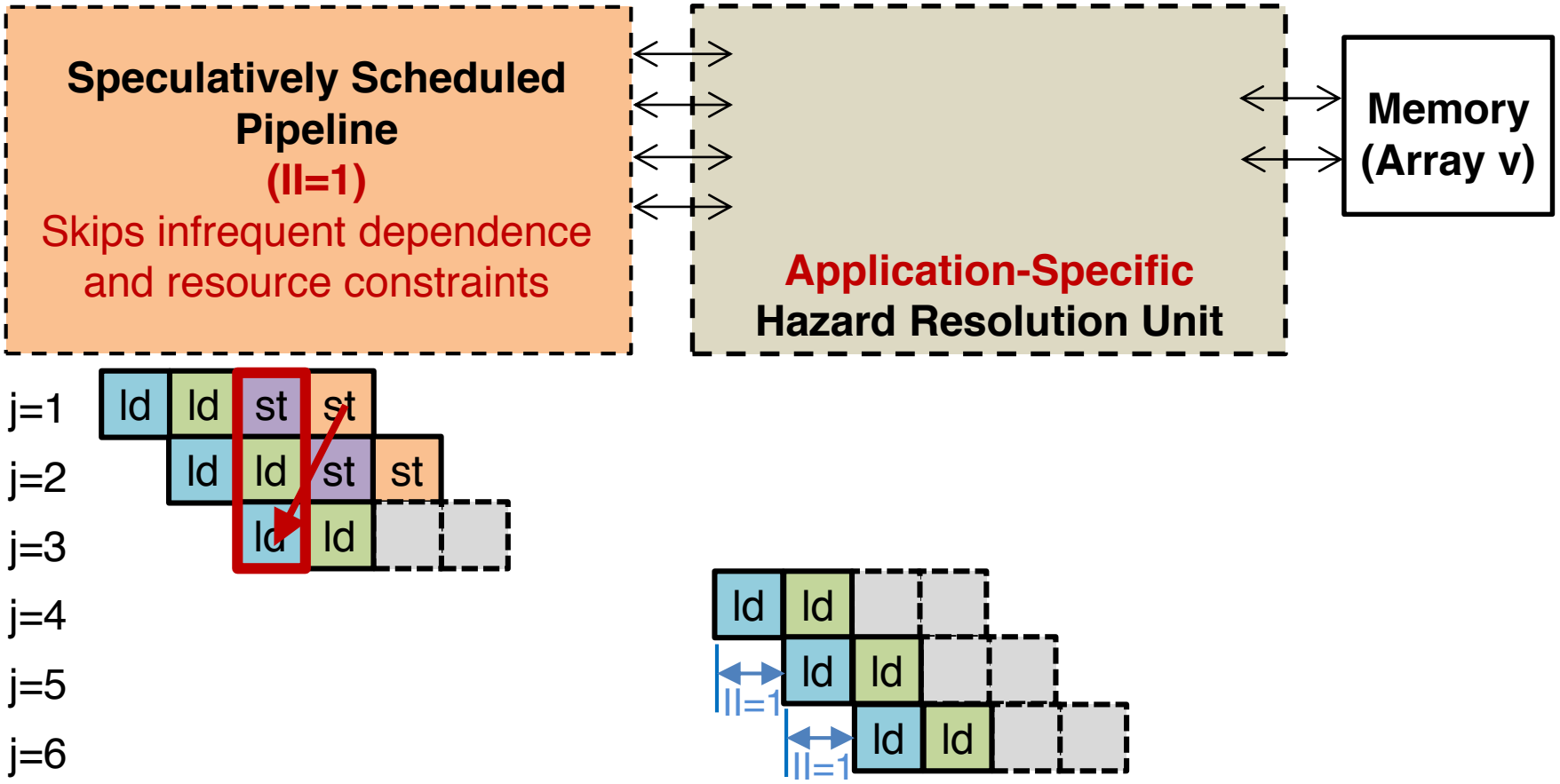
Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime



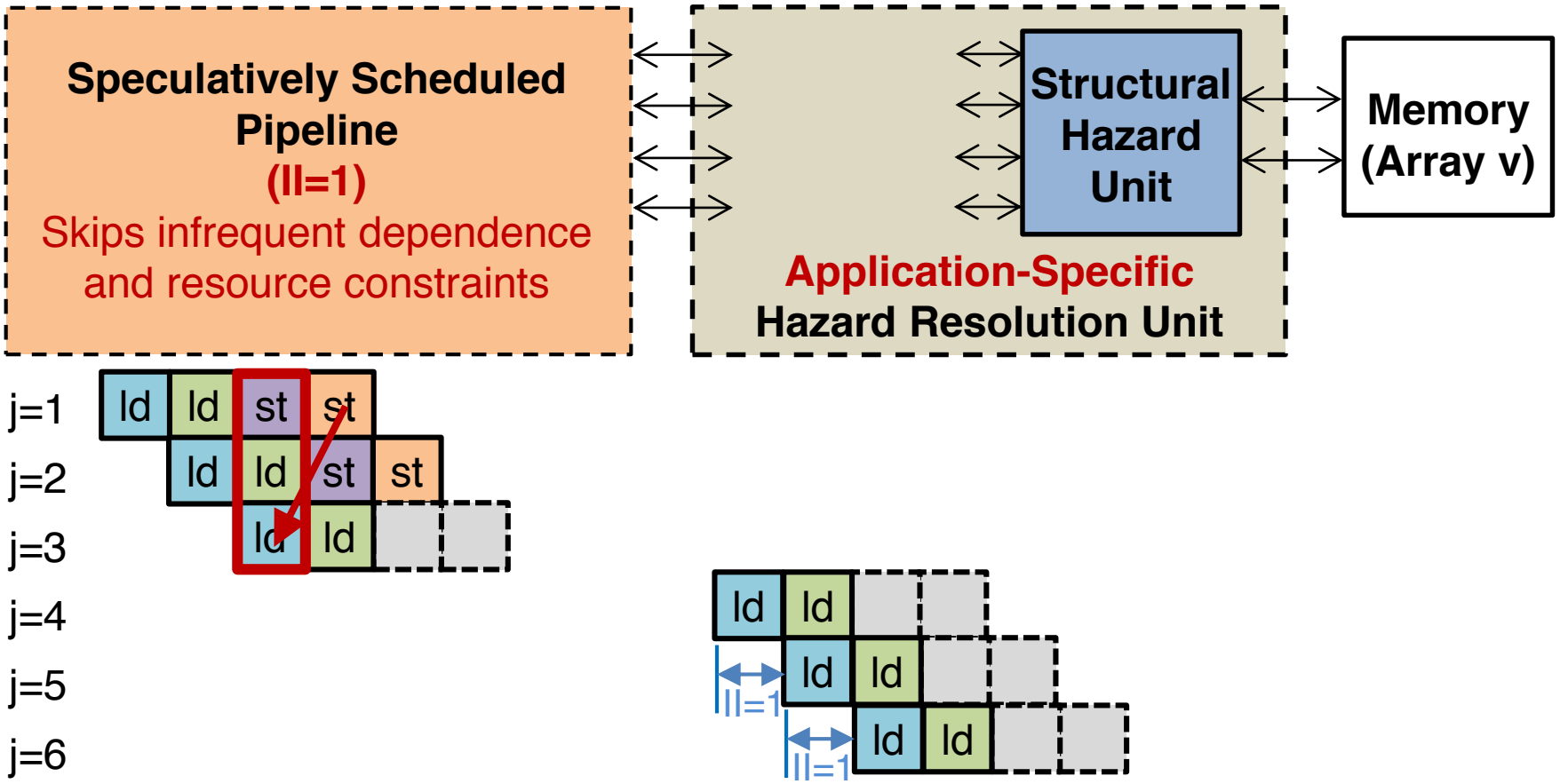
Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime



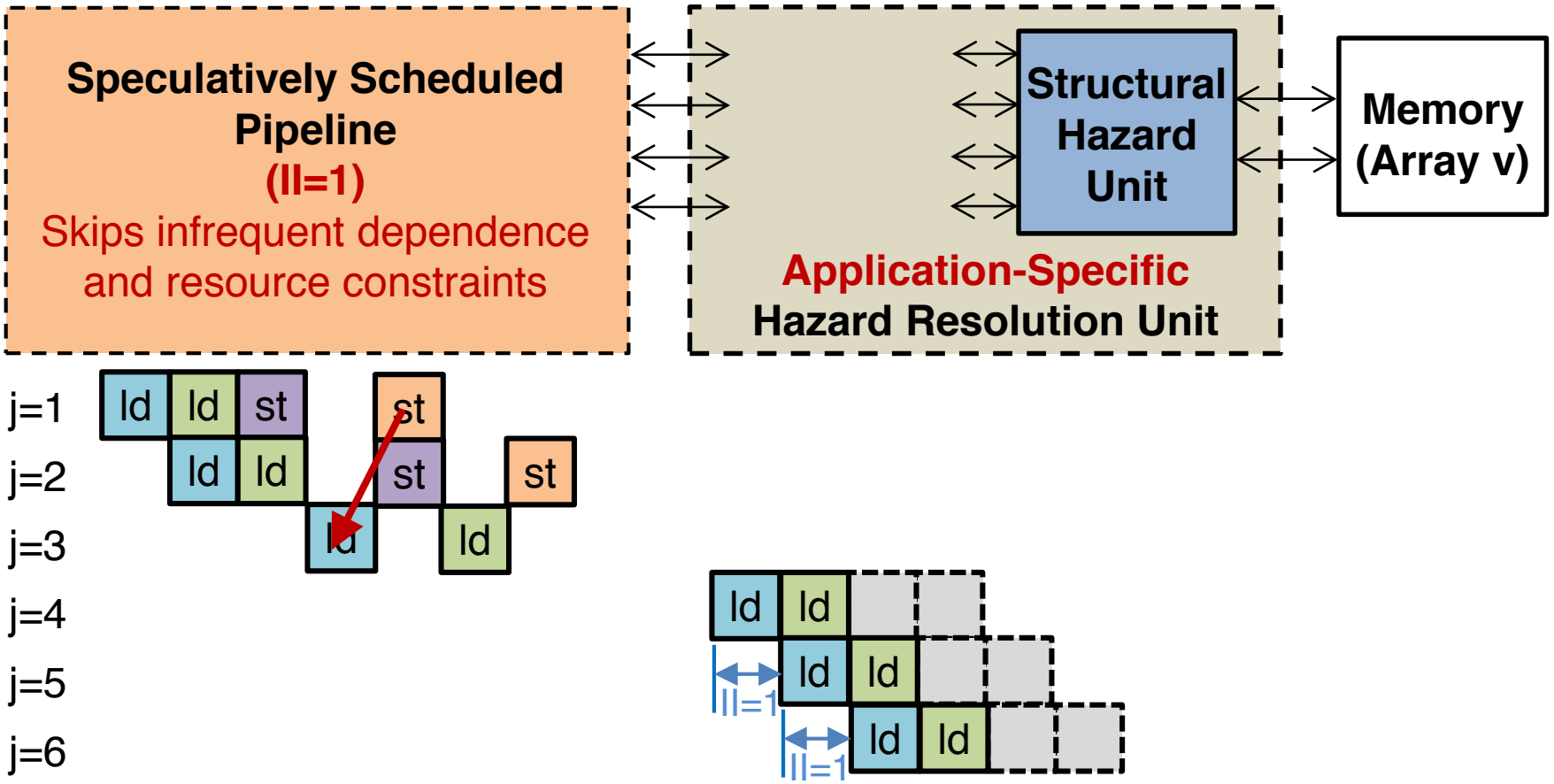
Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime



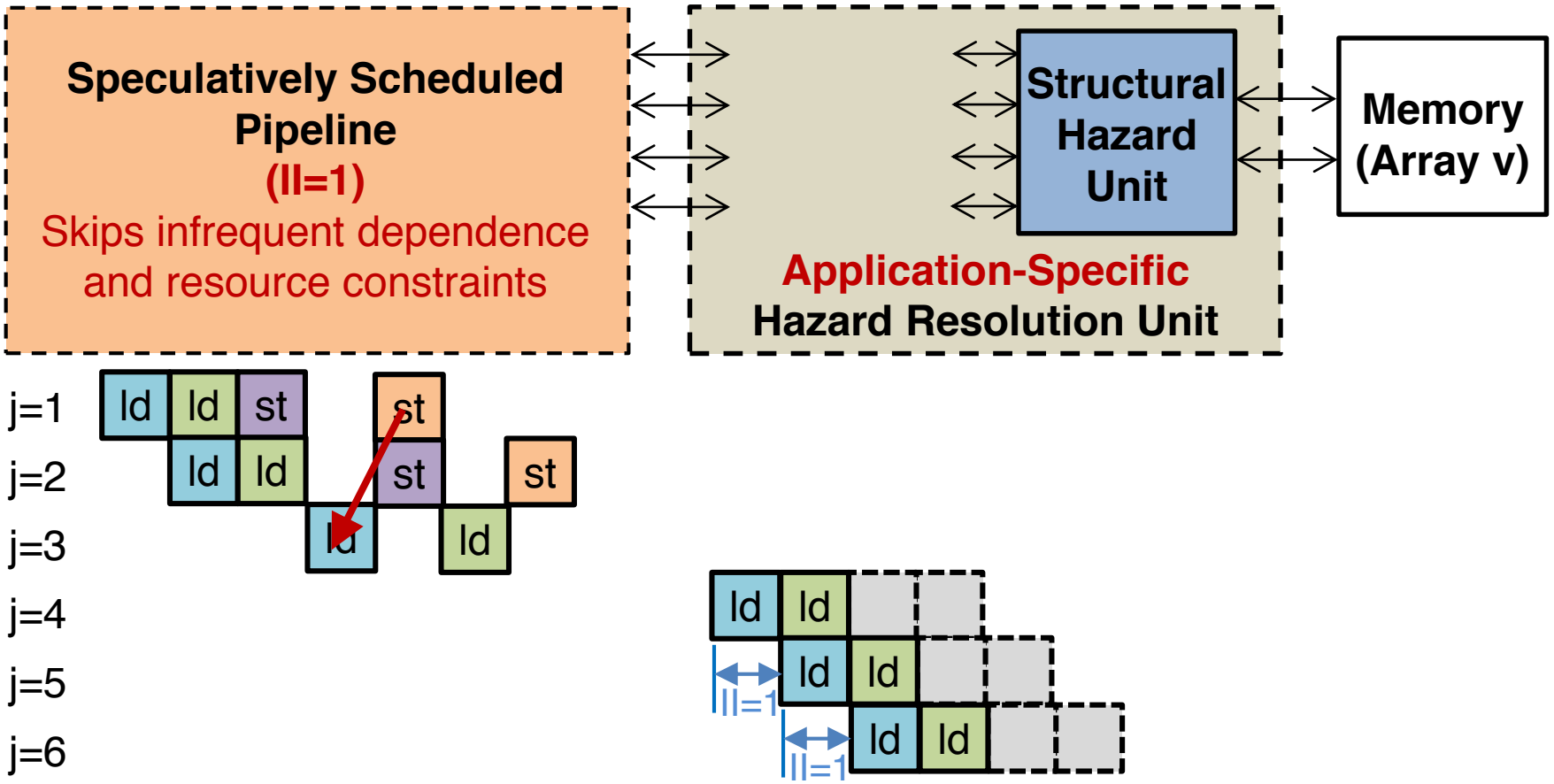
Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime



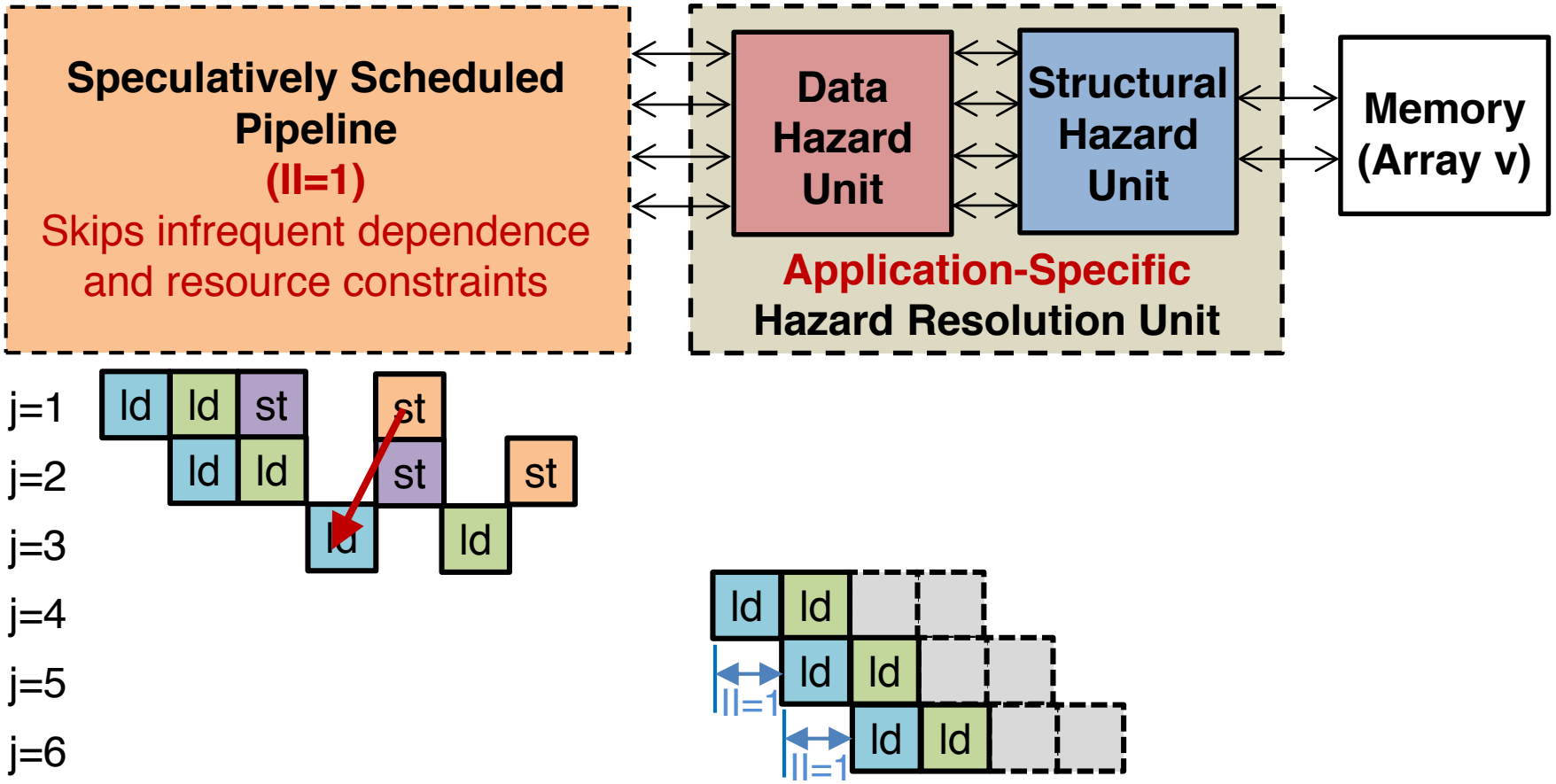
Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime



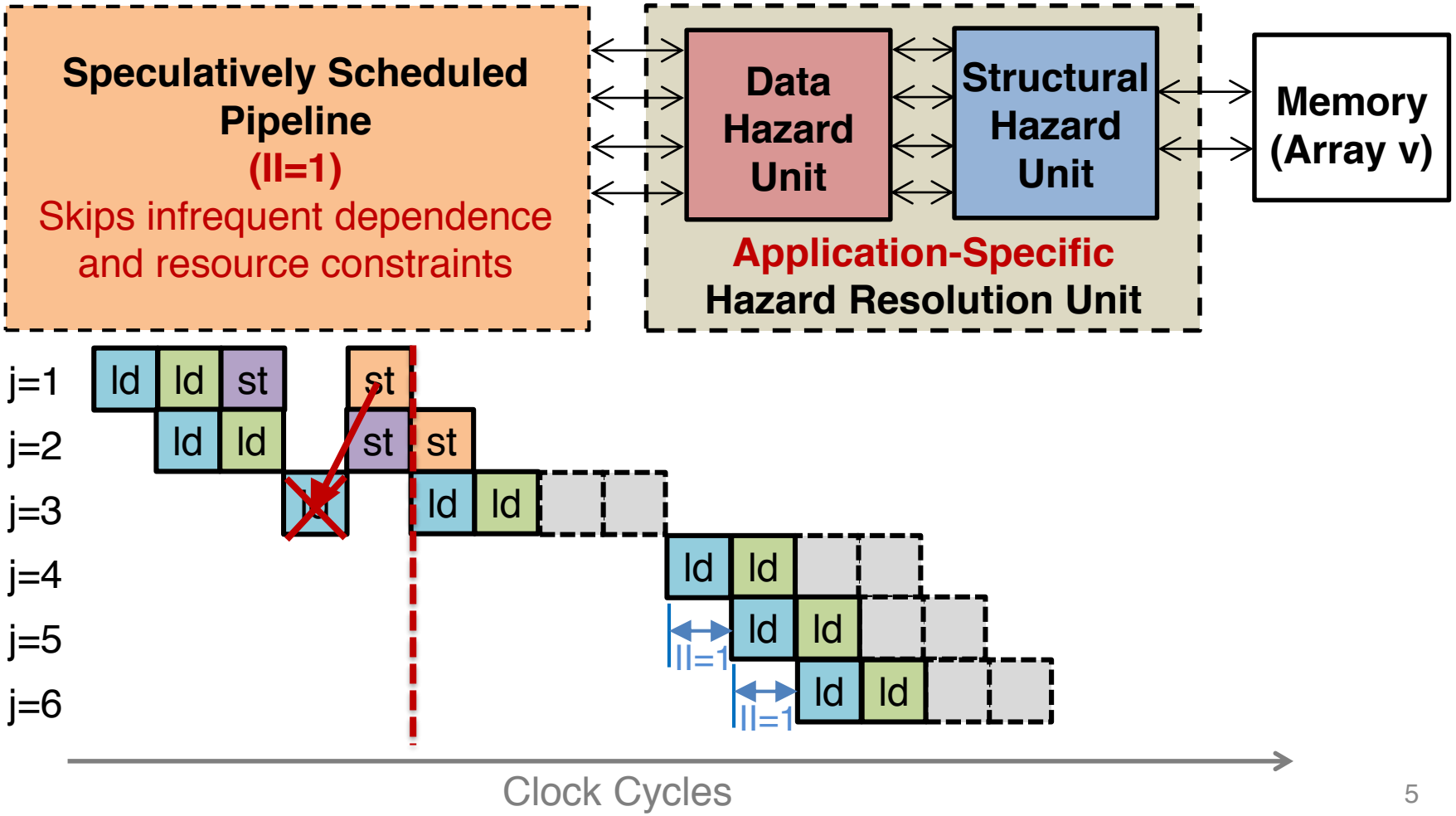
Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime



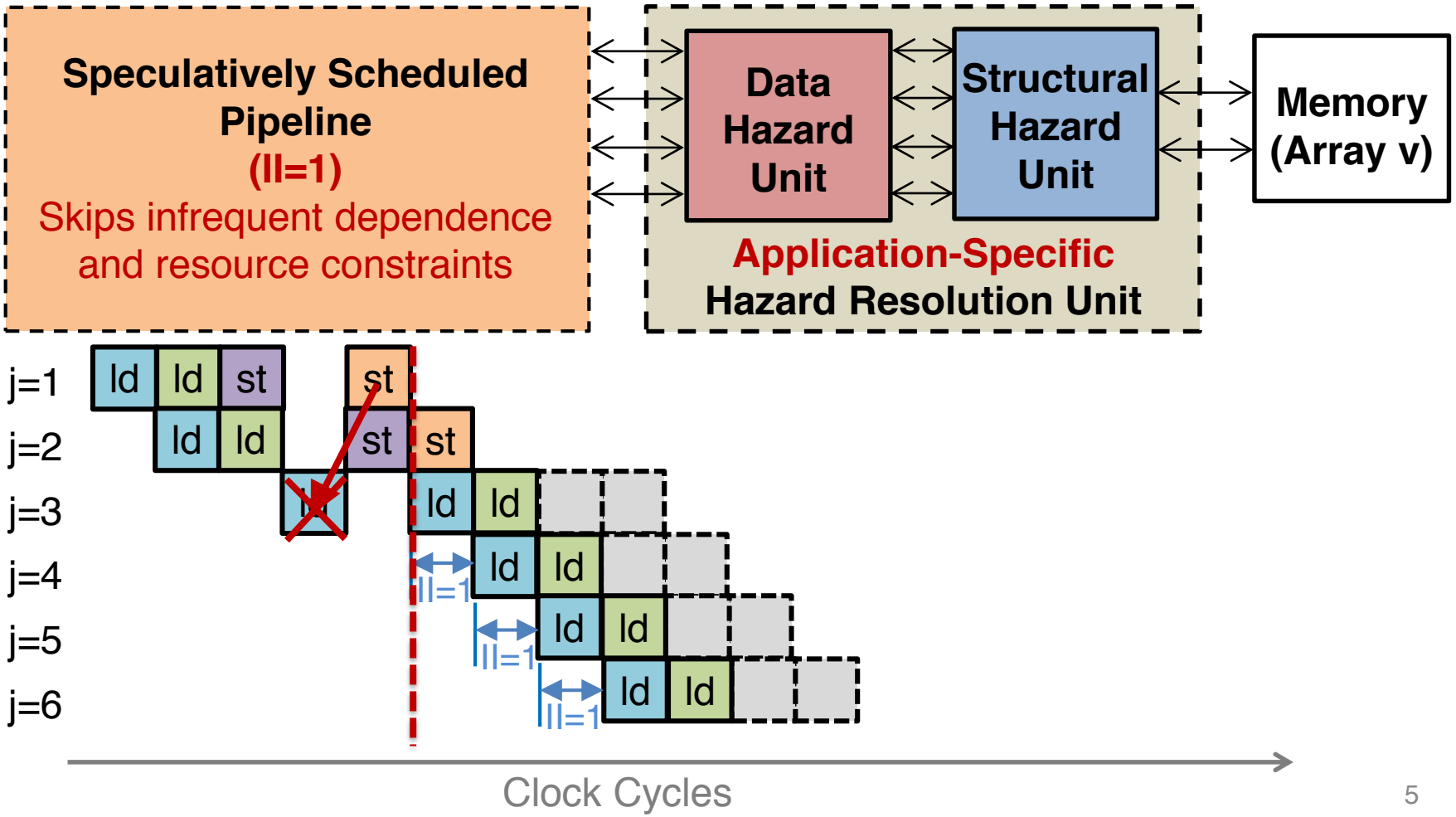
Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime



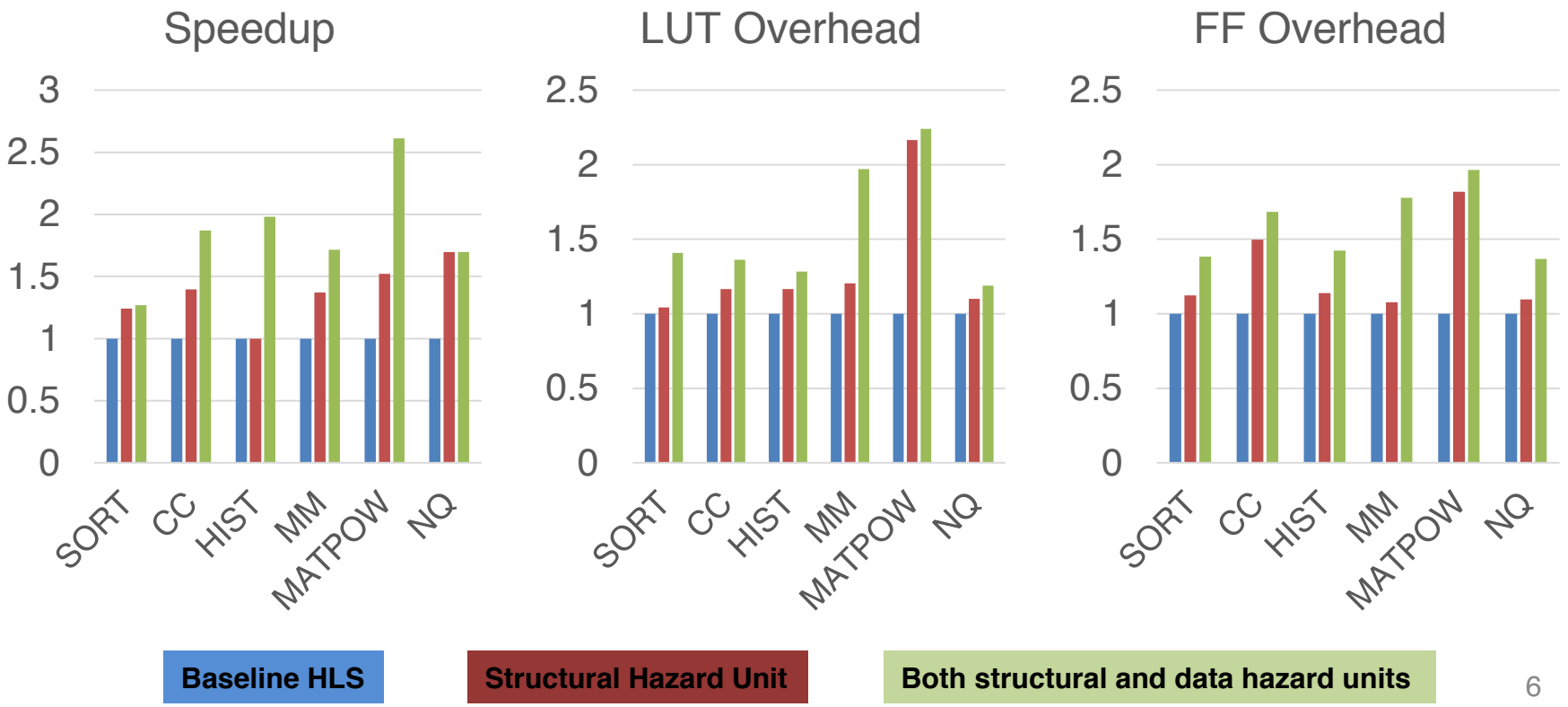
Dynamic Hazard Resolution for HLS

- Create a speculative pipeline for common case and rely on hazard resolution units to resolve hazards at runtime



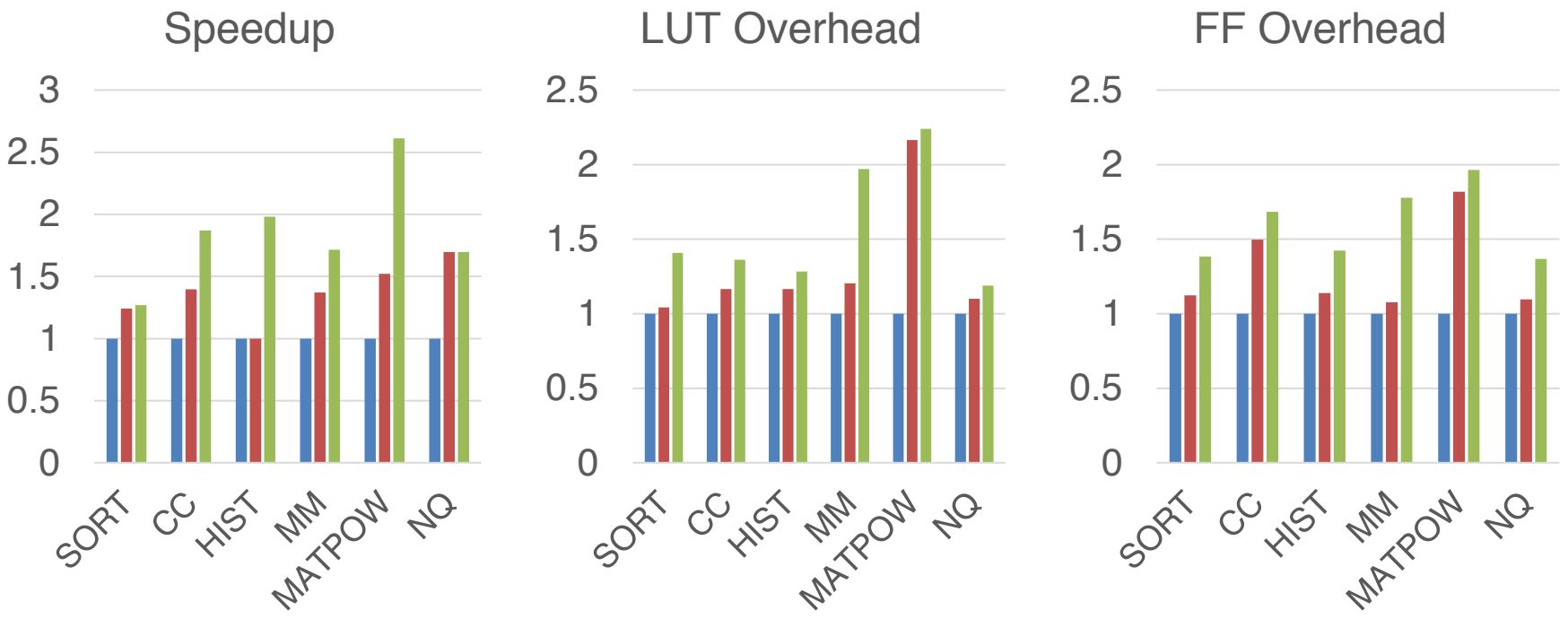
Experimental Results

- Significant speedup with reasonable area overhead
  - Amount of speedup depends on input pattern, data access pattern, and available memory bandwidth
  - Tradeoff between performance gain and hazard logic overhead



Experimental Results

- Significant speedup with reasonable area overhead
  - Amount of speedup depends on input pattern, data access pattern, and available memory bandwidth
  - Tradeoff between performance gain and hazard logic overhead



Experimental Results

- Significant speedup with reasonable area overhead
  - Amount of speedup depends on input pattern, data access pattern, and available memory bandwidth
  - Tradeoff between performance gain and hazard logic overhead