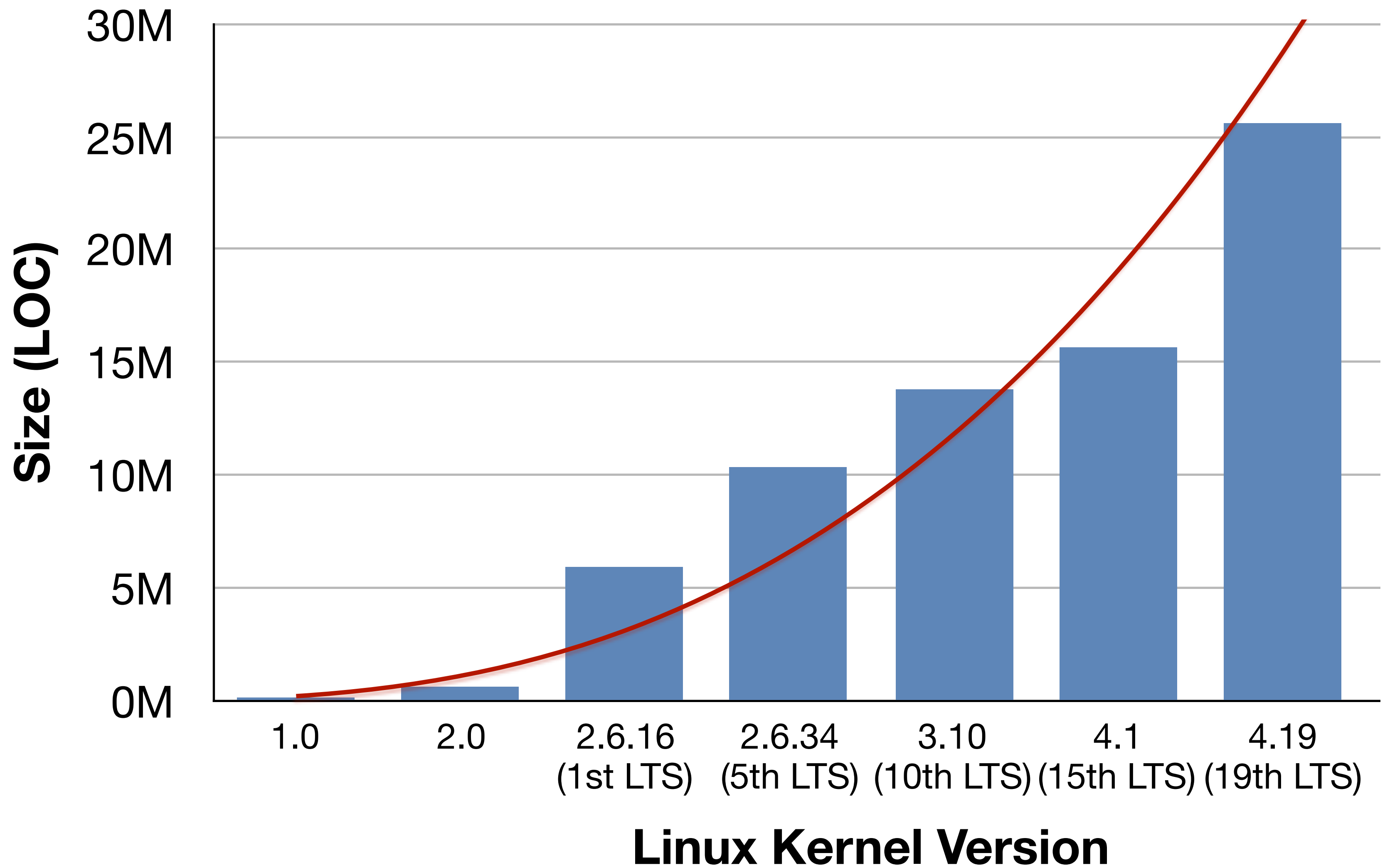
Resource-aware Program Analysis via Online Abstraction Coarsening

Kihong Heo
Hakjoo Oh
Hongseok Yang

ICSE 2019

Motivation

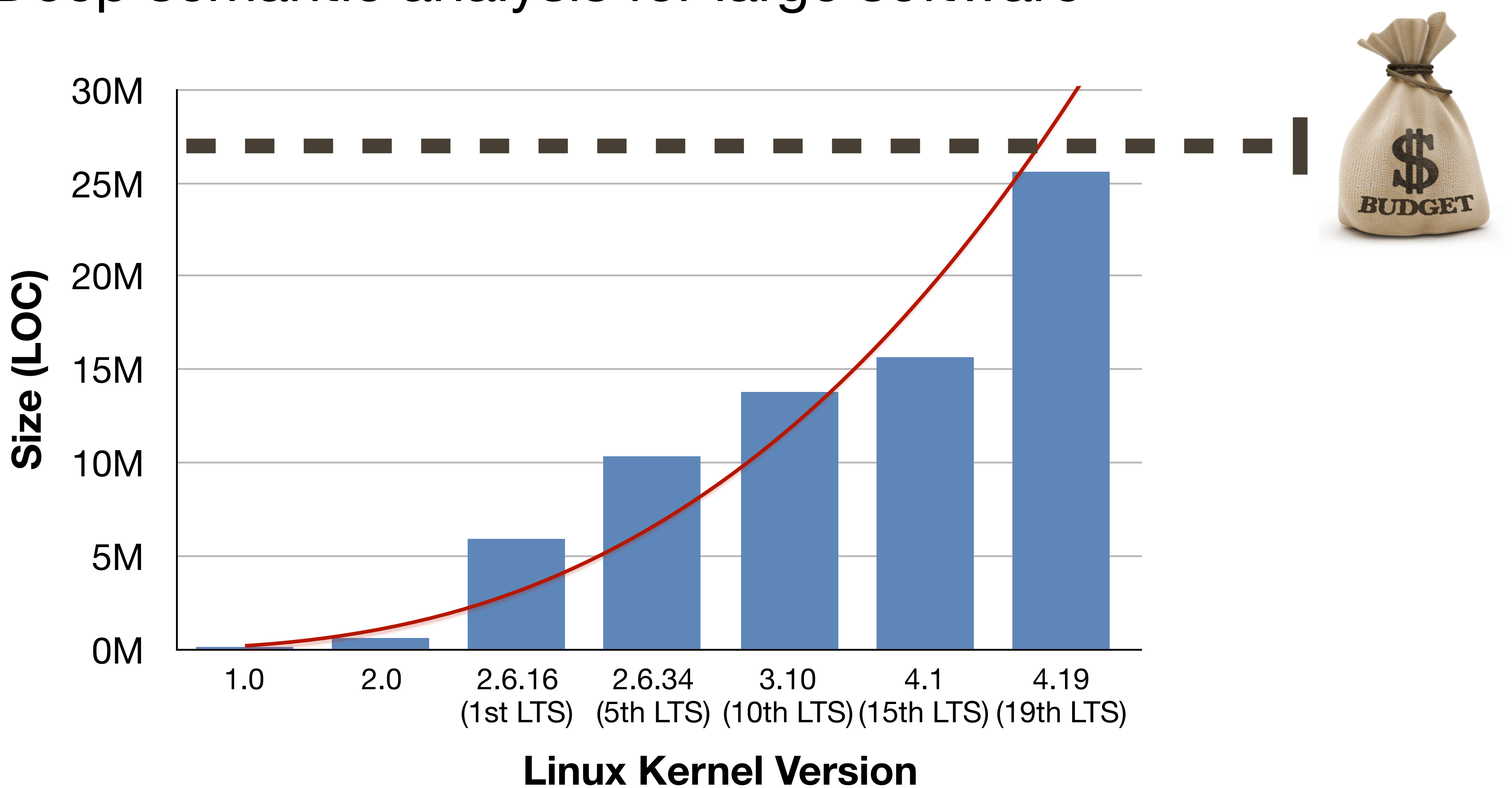
- Deep semantic analysis for large software



*https://www.linuxcounter.net

Motivation

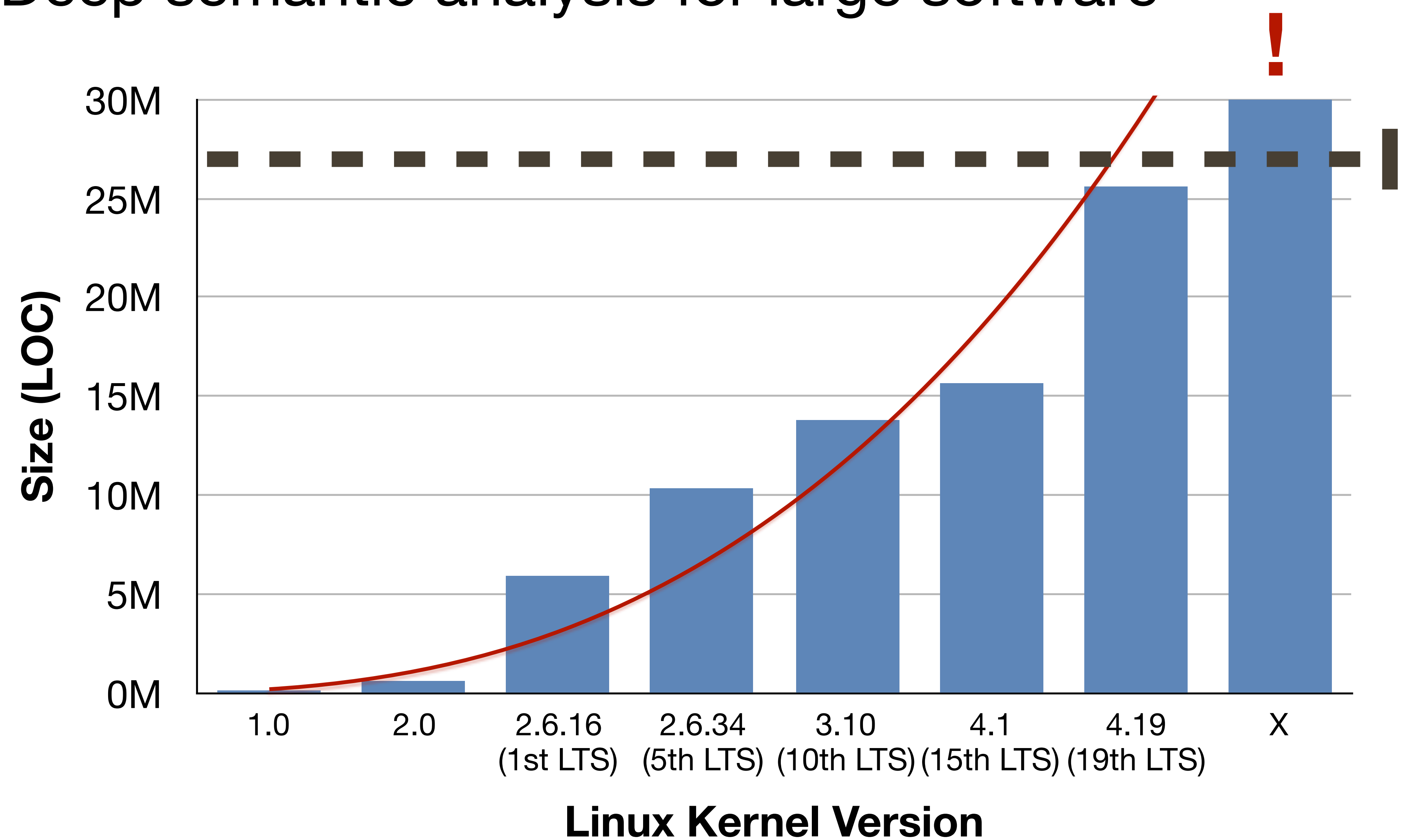
- Deep semantic analysis for large software



*https://www.linuxcounter.net

Motivation

- Deep semantic analysis for large software



*https://www.linuxcounter.net

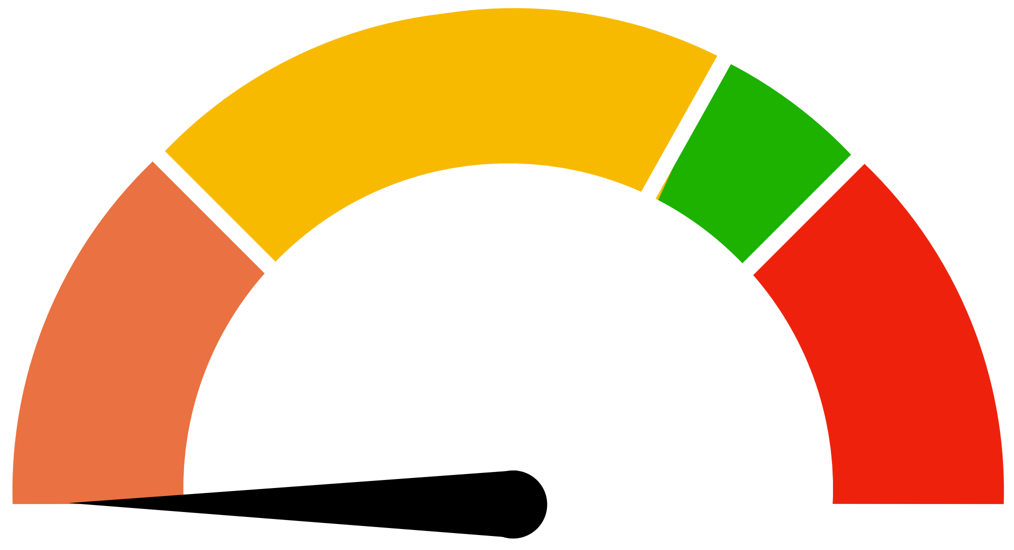
Goal

Goal

- Achieving **maximum precision** within a given **resource budget**
  - e.g., within 128GB of memory

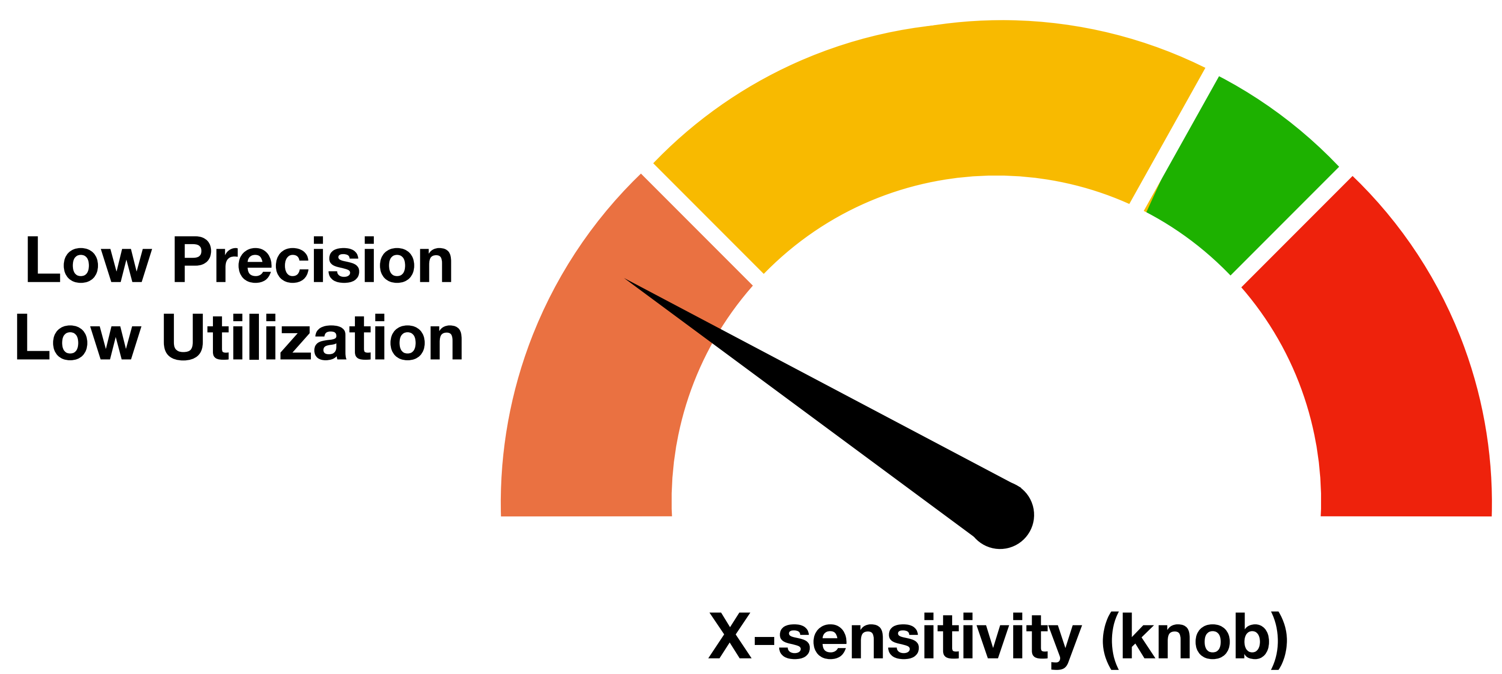
Goal

- Achieving **maximum precision** within a given **resource budget**
    - e.g., within 128GB of memory



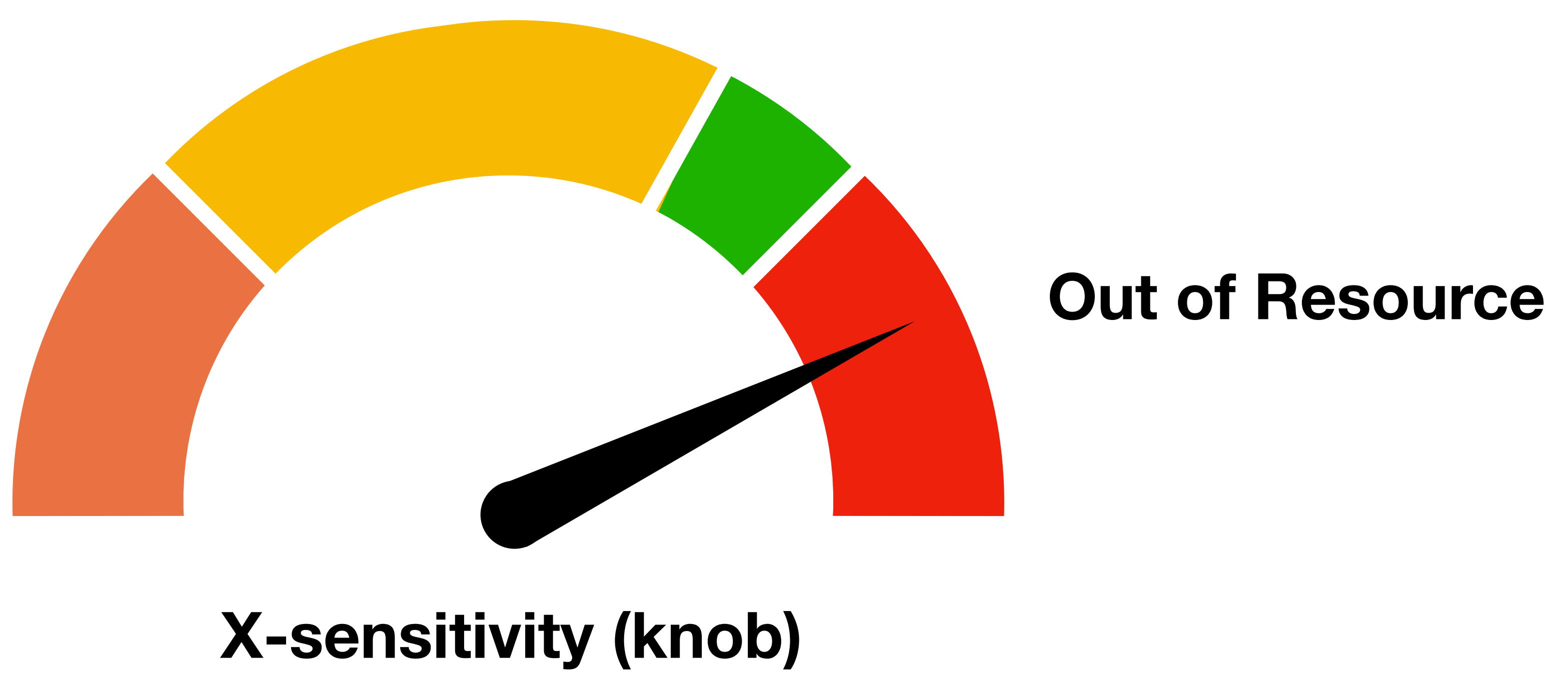
Goal

- Achieving **maximum precision** within a given **resource budget**
  - e.g., within 128GB of memory



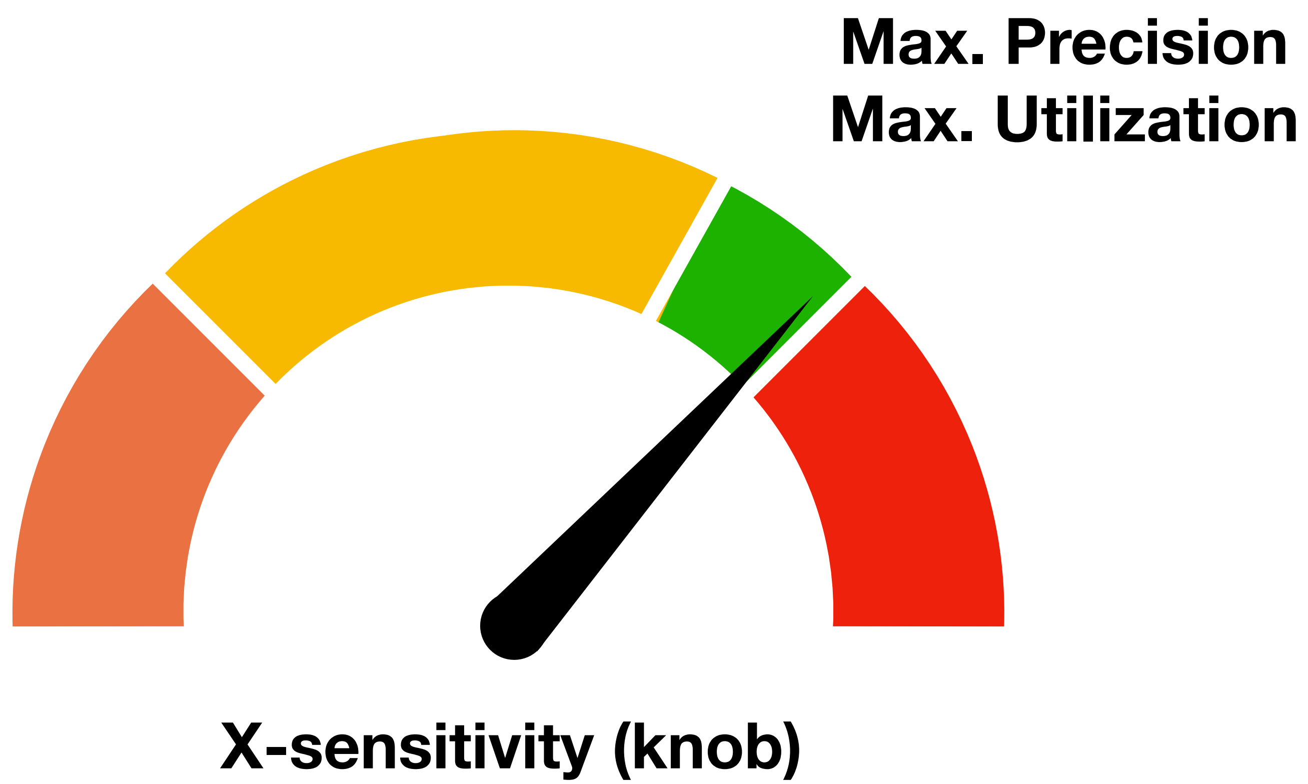
Goal

- Achieving **maximum precision** within a given **resource budget**
  - e.g., within 128GB of memory



Goal

- Achieving **maximum precision** within a given **resource budget**
  - e.g., within 128GB of memory



Challenges

- Hard to predict the behavior of analyzer in advance
  - e.g., partially flow-sensitive interval analysis

Challenges

- Hard to predict the behavior of analyzer in advance
  - e.g., partially flow-sensitive interval analysis

Sensitivity: 0%
emacs-26.0.91
(503KLOC)

Memory:
18GB

Challenges

- Hard to predict the behavior of analyzer in advance
    - e.g., partially flow-sensitive interval analysis

Sensitivity: 0% emacs-26.0.91 (503KLOC)	$<$	Sensitivity: 5% emacs-26.0.91 (503KLOC)

Memory:
18GB

Challenges

- Hard to predict the behavior of analyzer in advance
  - e.g., partially flow-sensitive interval analysis

Sensitivity: 0% emacs-26.0.91 (503KLOC)	<	Sensitivity: 5% emacs-26.0.91 (503KLOC)
Memory: 18GB	<<	Memory: > 128GB

Challenges

- Hard to predict the behavior of analyzer in advance
  - e.g., partially flow-sensitive interval analysis

Sensitivity: 0% vim60 (227KLOC)	<	Sensitivity: 0% emacs-26.0.91 (503KLOC)	<	Sensitivity: 5% emacs-26.0.91 (503KLOC)
		Memory: 18GB	<<	Memory: > 128GB

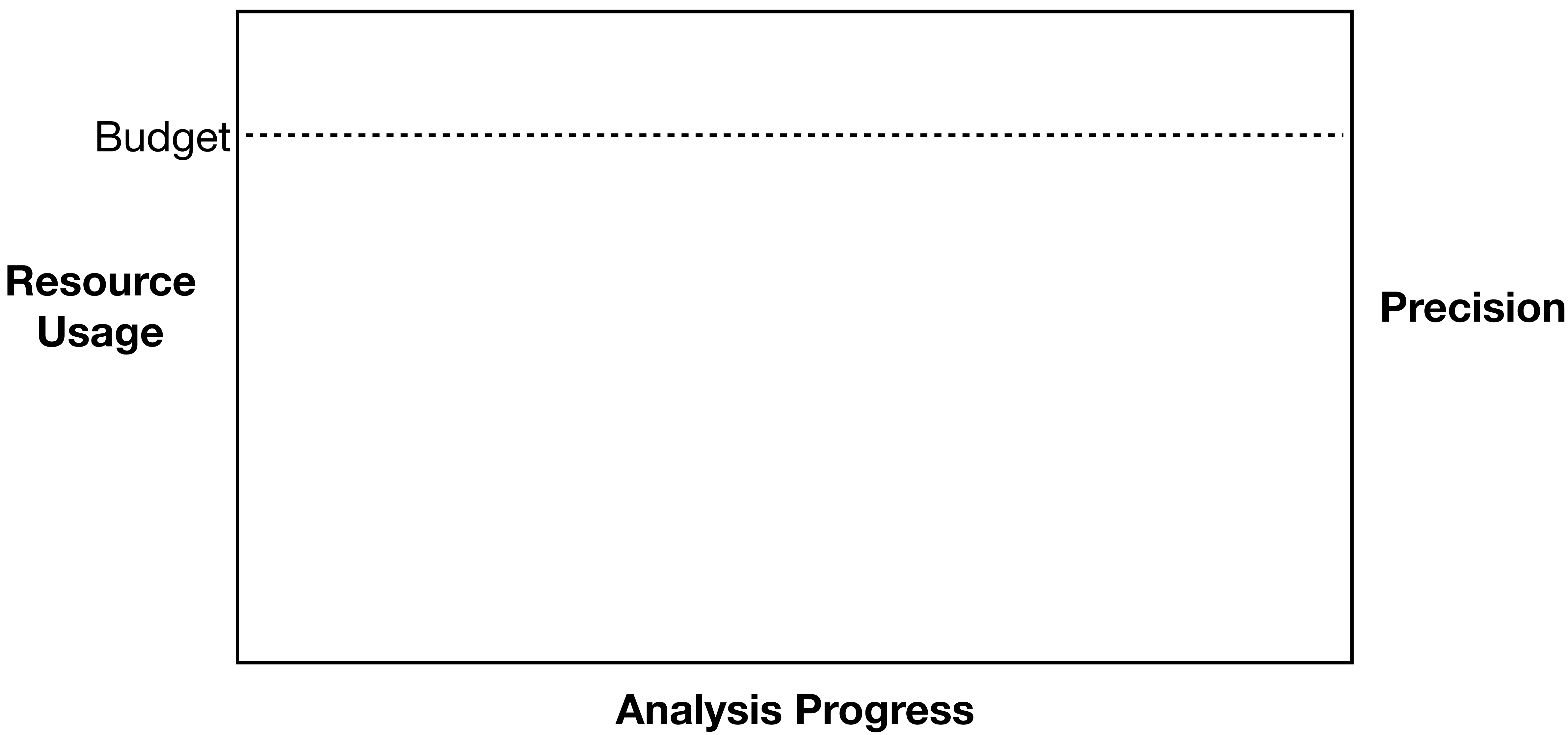
Challenges

- Hard to predict the behavior of analyzer in advance
  - e.g., partially flow-sensitive interval analysis

Sensitivity: 0% vim60 (227KLOC)	<	Sensitivity: 0% emacs-26.0.91 (503KLOC)	<	Sensitivity: 5% emacs-26.0.91 (503KLOC)
Memory: 51GB	>	Memory: 18GB	<<	Memory: > 128GB

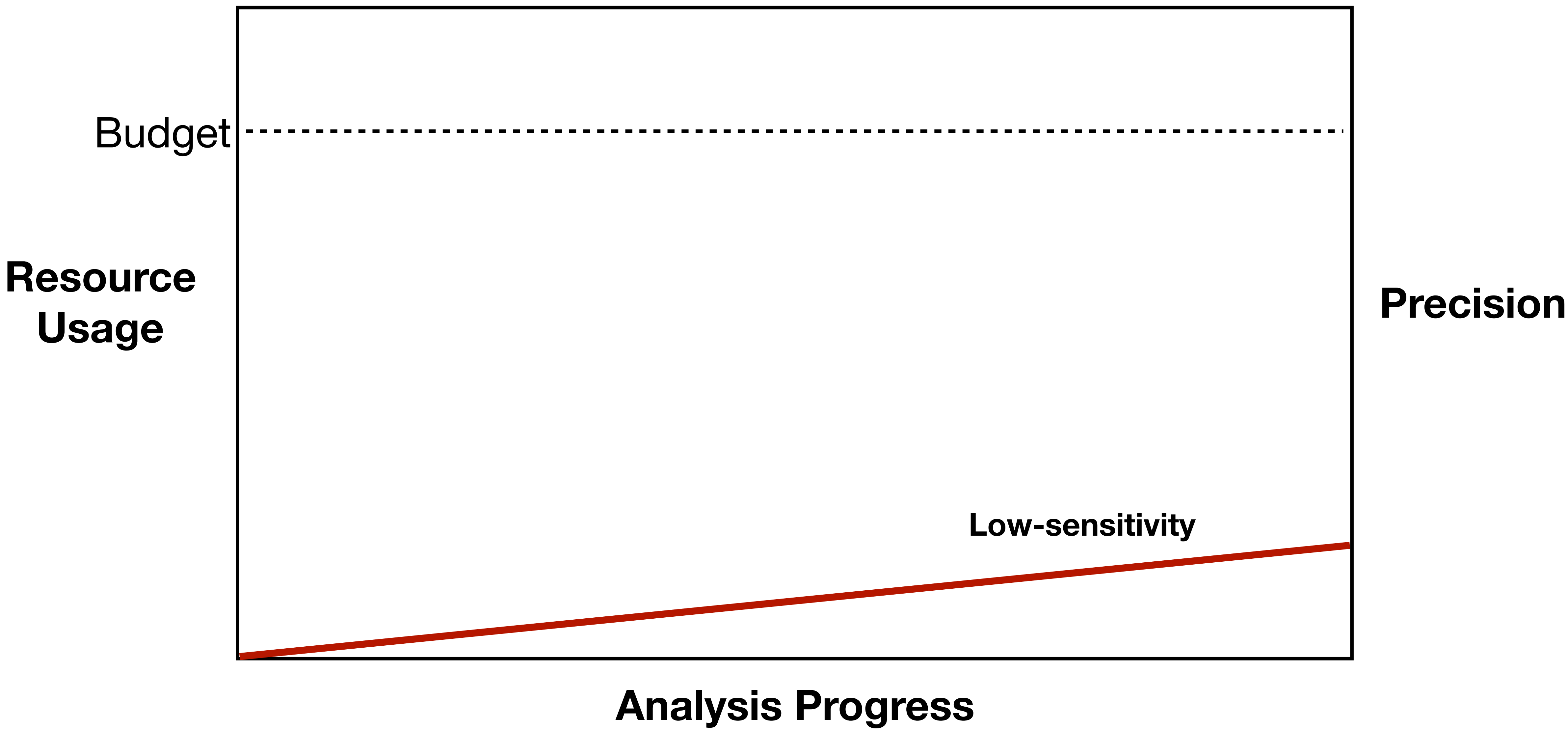
Our Approach

- Online abstraction coarsening by a learned controller



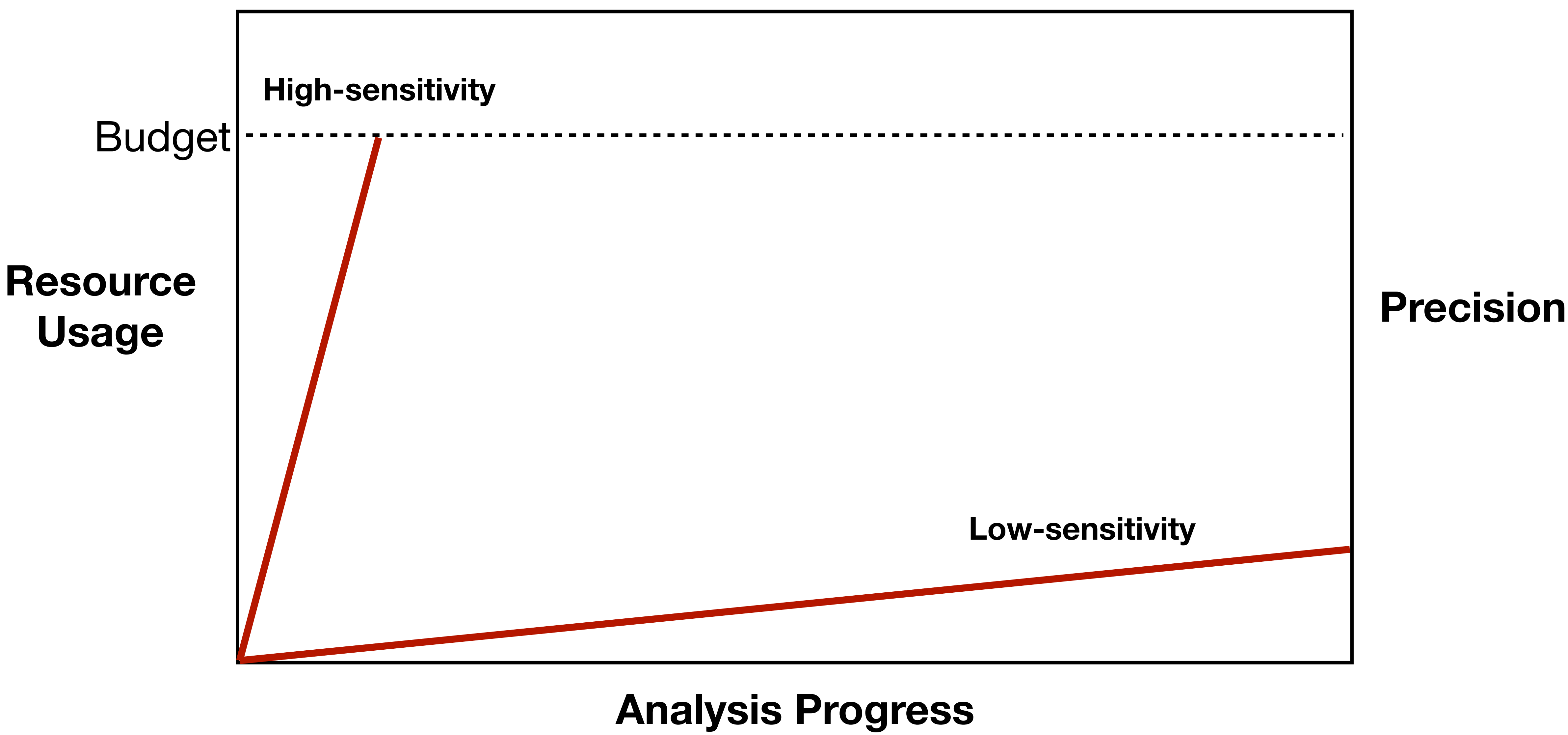
Our Approach

- Online abstraction coarsening by a learned controller



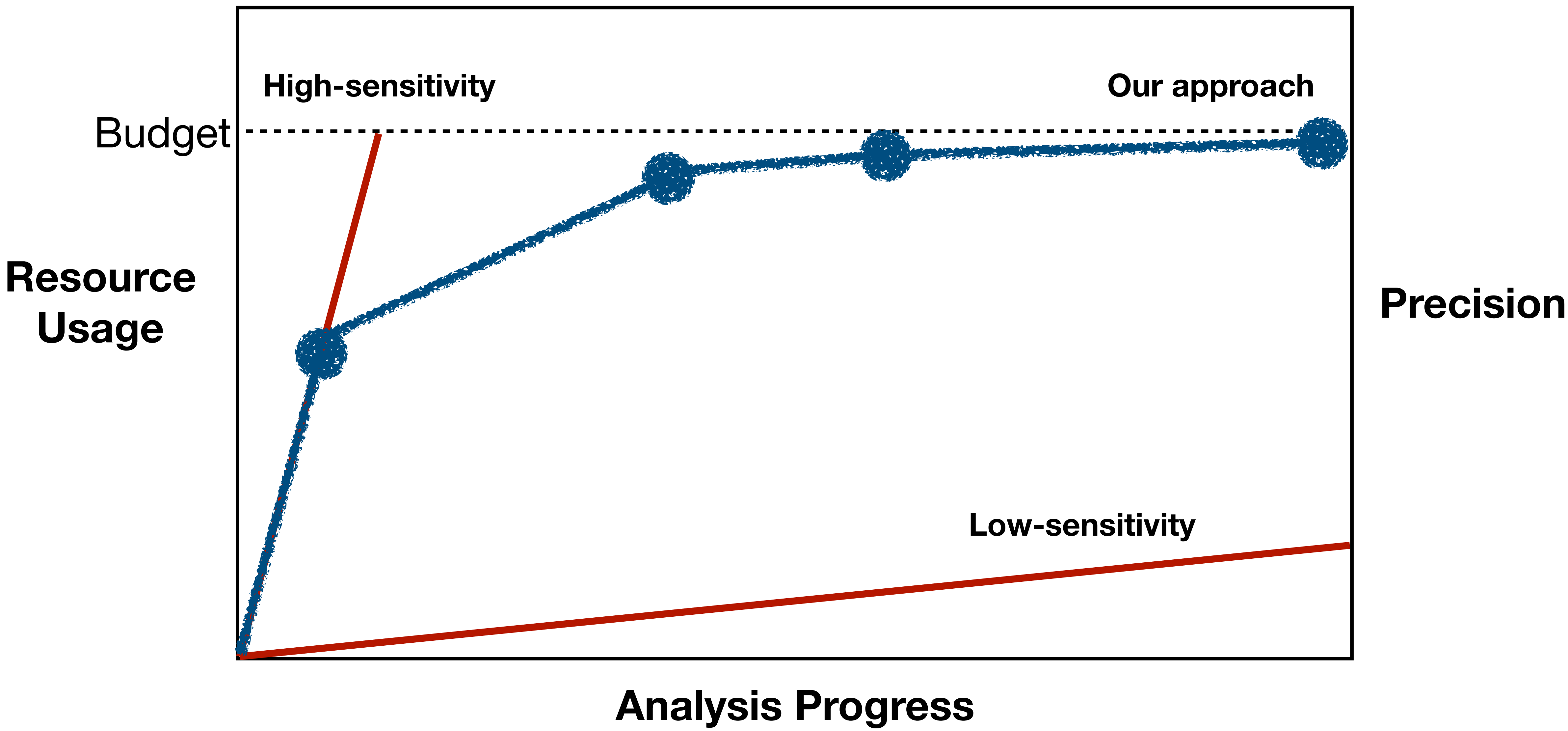
Our Approach

- Online abstraction coarsening by a learned controller



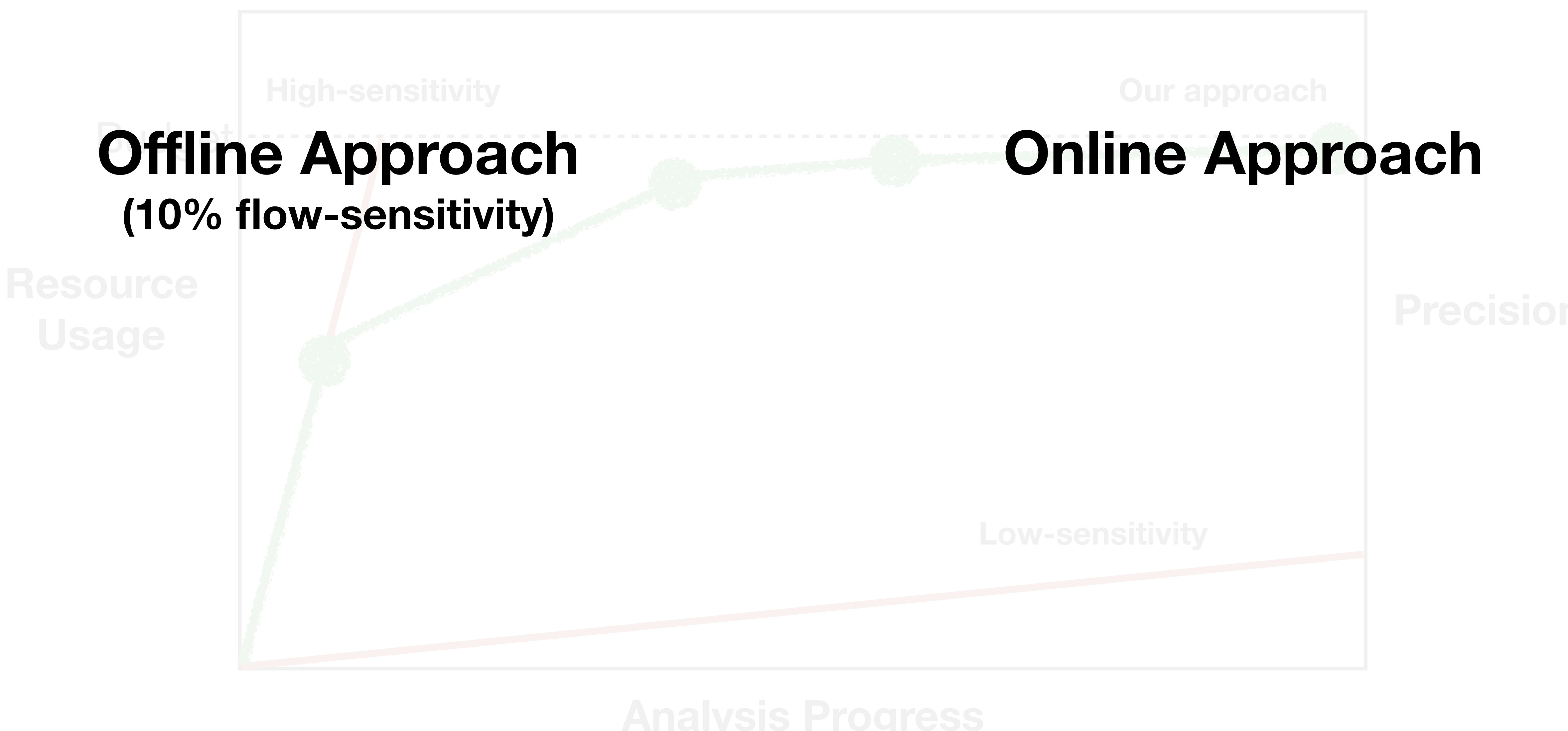
Our Approach

- Online abstraction coarsening by a learned controller



Our Approach

- Online abstraction coarsening by a learned controller



Offline Approach
(10% flow-sensitivity)

Online Approach

Our Approach

- Online abstraction coarsening by a learned controller

Offline Approach
(10% flow-sensitivity)

Online Approach

- **3/8** run out of memory (128GB)
- **27%** of buffer overrun alarms ↓
- **30%** of null dereference alarms ↓

Our Approach

- Online abstraction coarsening by a learned controller

Offline Approach
(10% flow-sensitivity)

- **3/8** run out of memory (128GB)
- **27%** of buffer overrun alarms ↓
- **30%** of null dereference alarms ↓

Online Approach

- **0/8** run out of memory (64 / 128GB)
- **28—32%** of buffer overrun alarms ↓
- **33—41%** of null dereference alarms ↓

Outline

- Motivation
- **Learning Framework**
- Experimental Results
- Conclusion

Example

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);   // Query 3 (may fail)
```

Example

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);   // Query 3 (may fail)
```

Line	Flow-Sensitive Abstract State
1	{x = [0,0], y = [0,0], z = [1,1], v = ⊤, w = ⊤}

3 Intervals

Example

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);     // Query 1 (hold)
6: assert(z > 0);     // Query 2 (hold)
7: assert(v == w);    // Query 3 (may fail)
```

Line	Flow-Sensitive Abstract State
1	{x = [0,0], y = [0,0], z = [1,1], v = ⊤, w = ⊤}
2	{x = [1,1], y = [0,0], z = [1,1], v = ⊤, w = ⊤}

6 Intervals

Example

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);   // Query 3 (may fail)
```

Line	Flow-Sensitive Abstract State
1	{x = [0,0], y = [0,0], z = [1,1], v = ⊤, w = ⊤}
2	{x = [1,1], y = [0,0], z = [1,1], v = ⊤, w = ⊤}
3	{x = [1,1], y = [0,0], z = [2,2], v = ⊤, w = ⊤}
4	{**x = [1,1], y = [1,1], z = [2,2]**, v = ⊤, w = ⊤}

12 Intervals

Example

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);   // Query 3 (may fail)
```

Line	Flow-Insensitive Abstract State
*	{**x = [0,+∞], y = [0,+∞], z = [1,+∞]**, v = ⊤, w = ⊤}

3 Intervals

Online Abstraction Coarsening

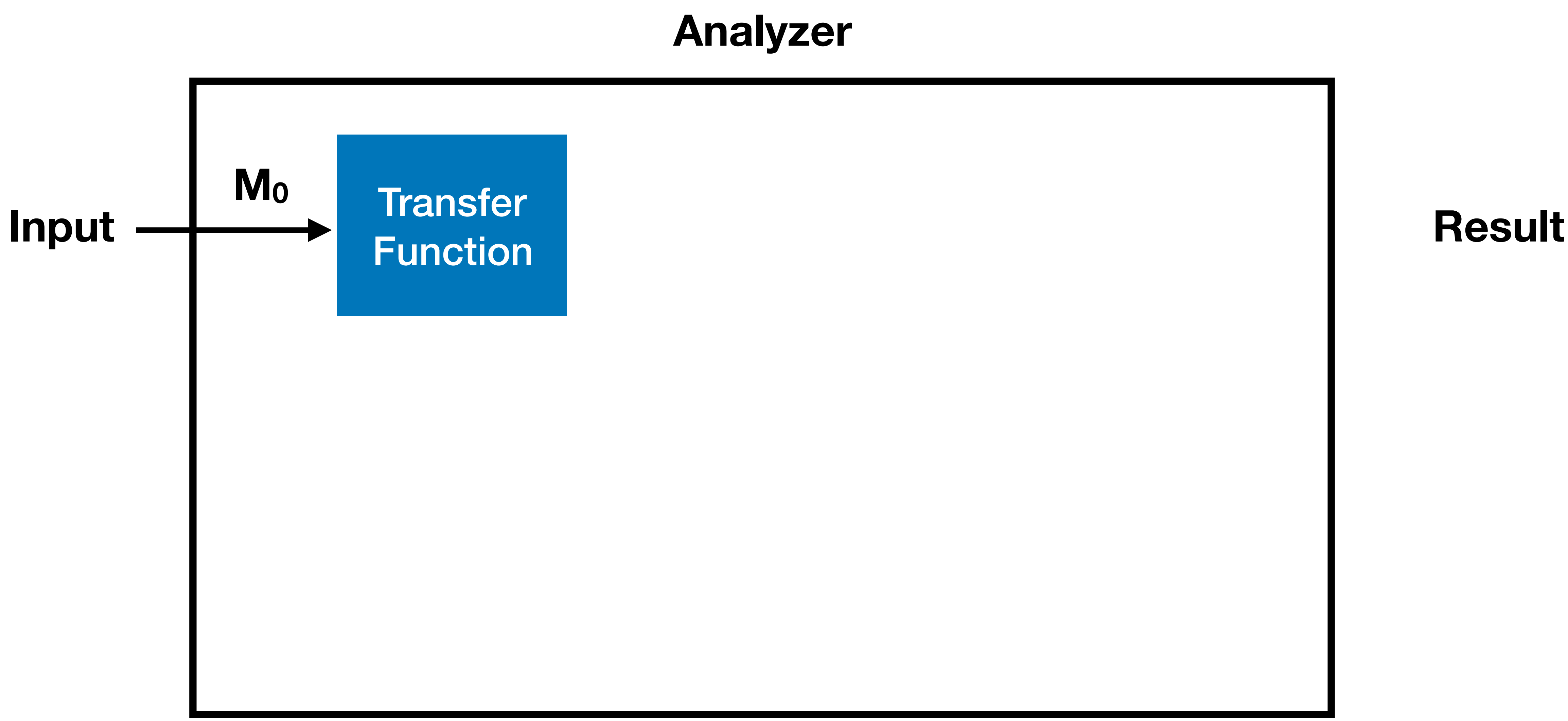
Online Abstraction Coarsening

Analyzer

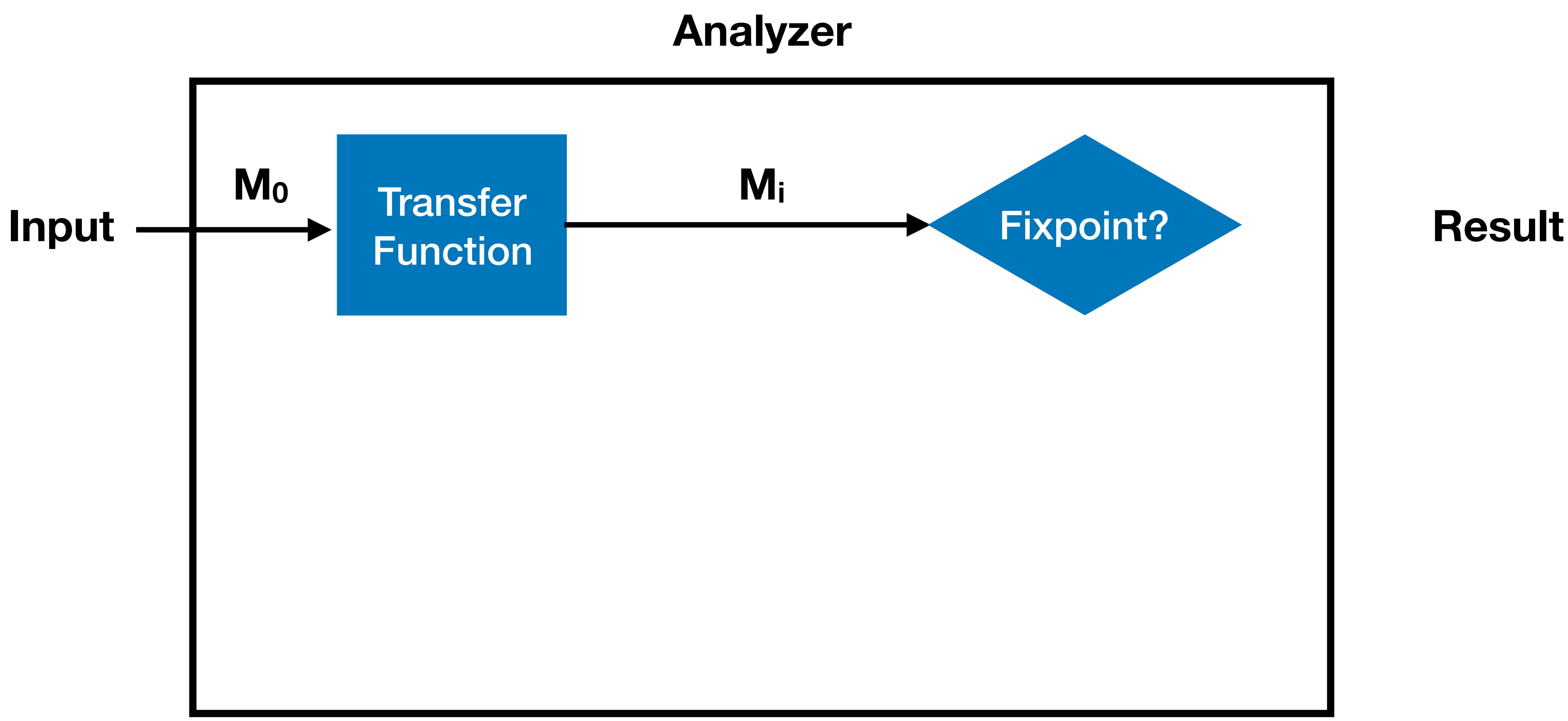
Input

Result

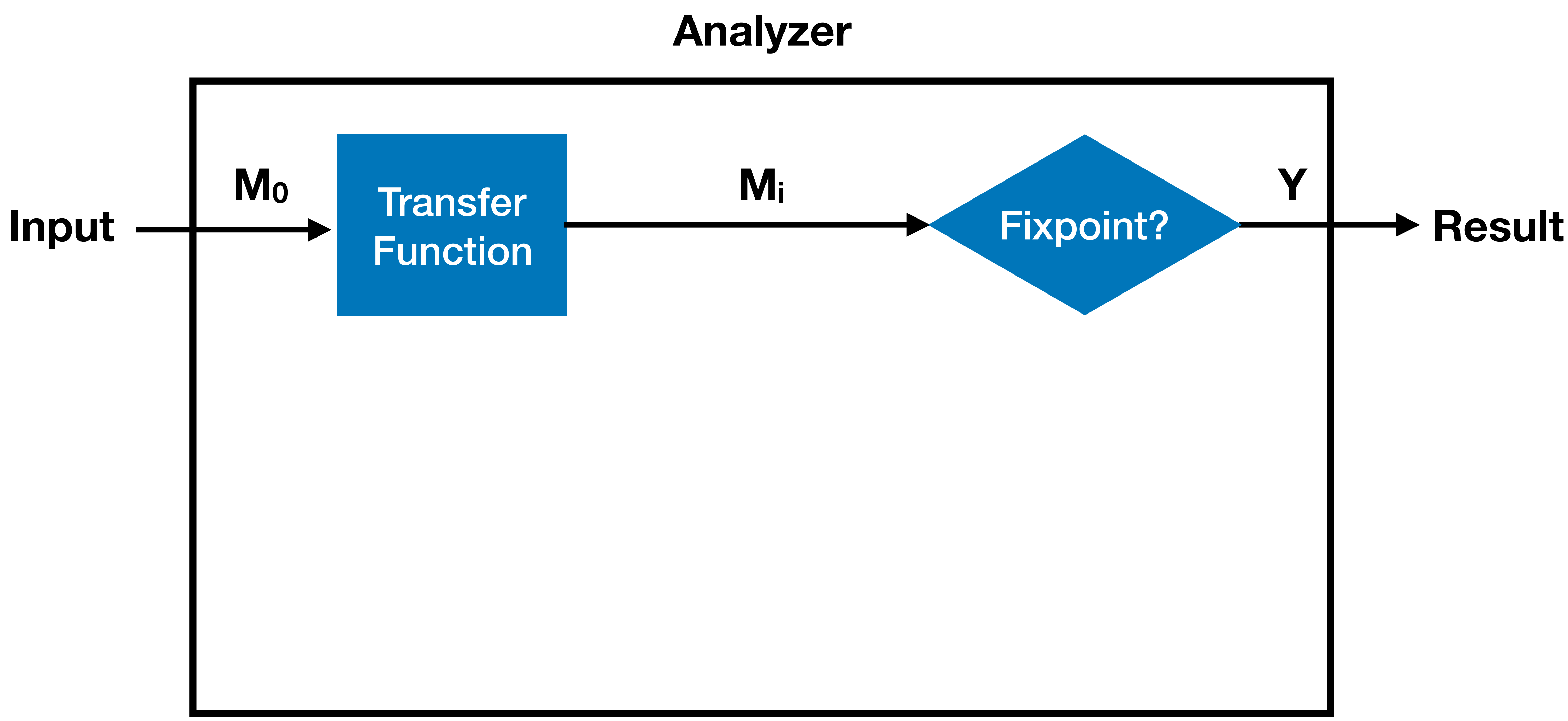
Online Abstraction Coarsening



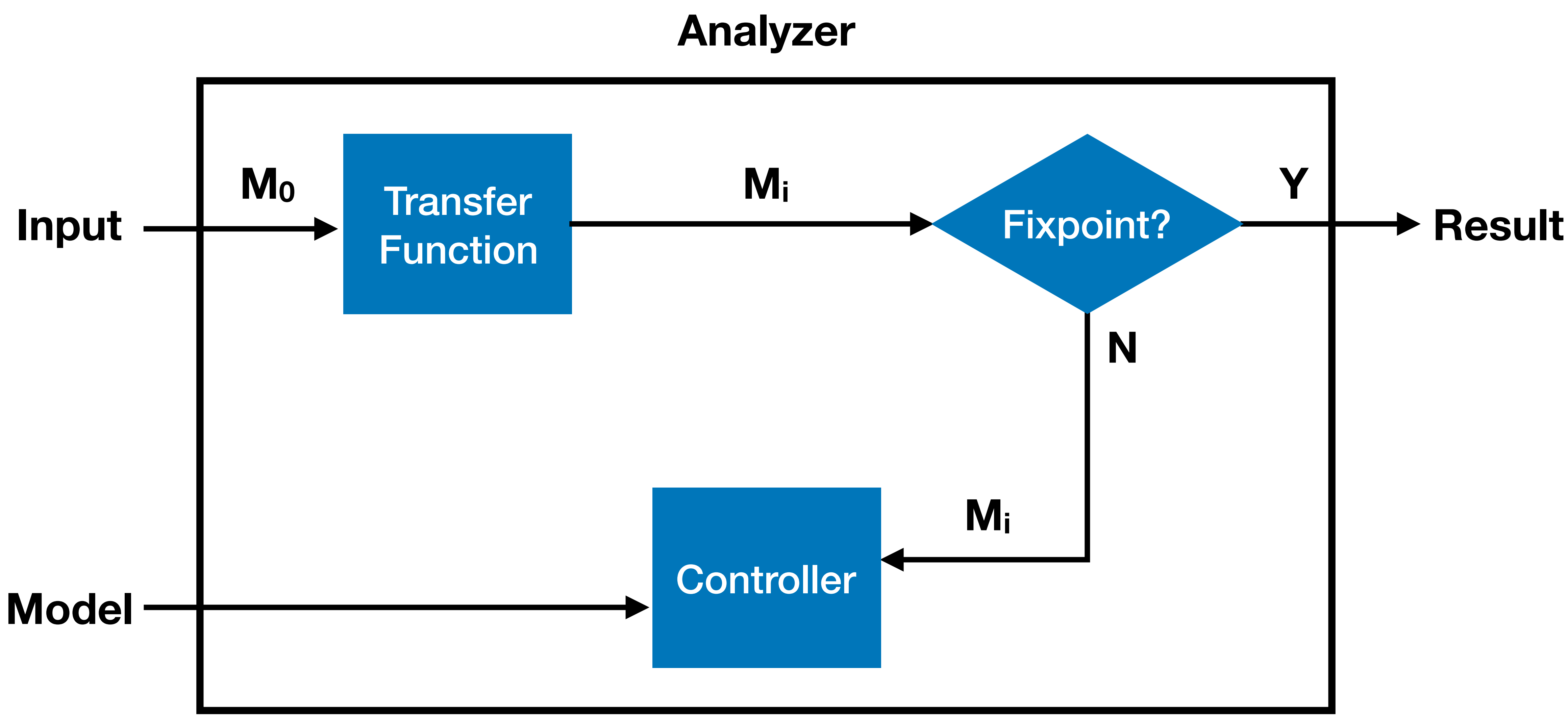
Online Abstraction Coarsening



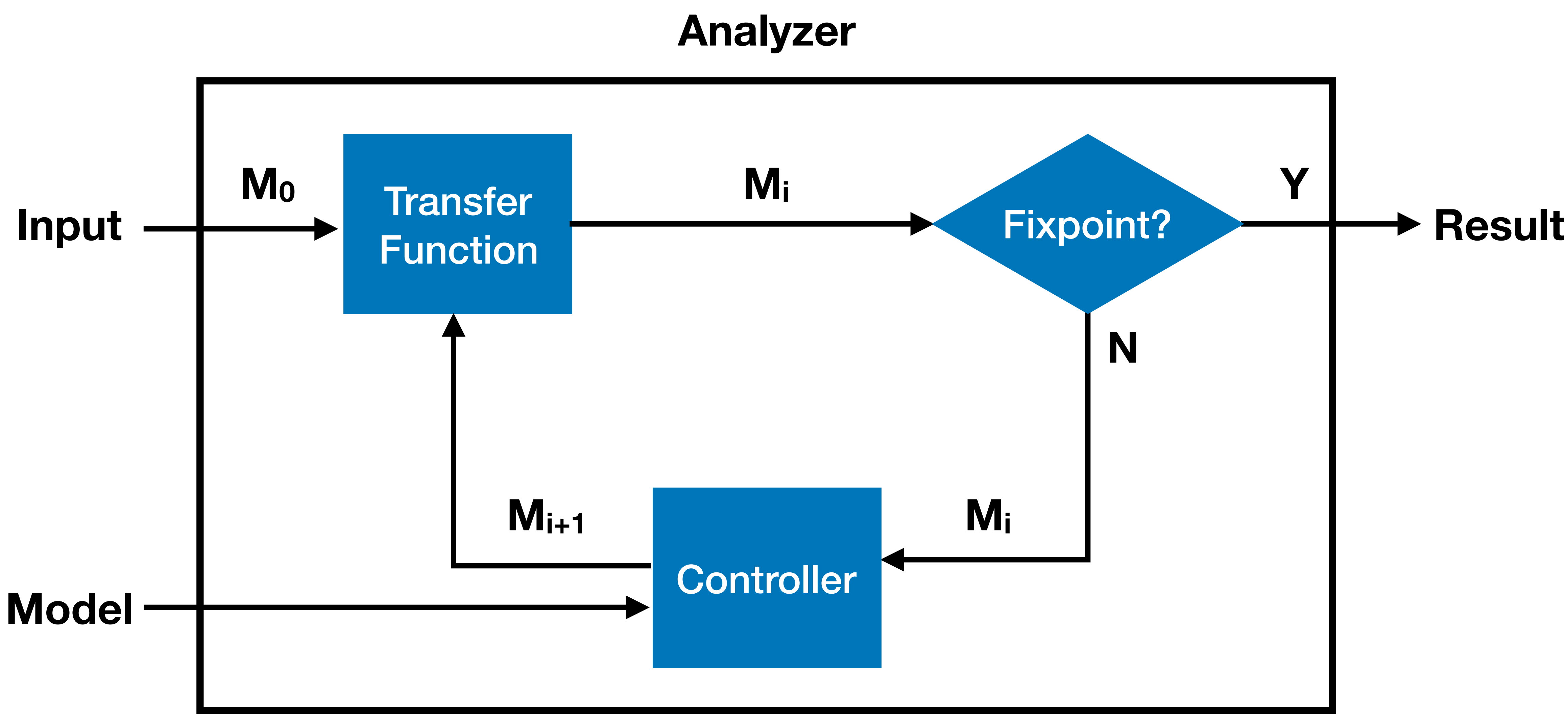
Online Abstraction Coarsening



Online Abstraction Coarsening



Online Abstraction Coarsening



Model

Model

- Model *M :* Variable $\rightarrow$ [0, 1]
- Importance of each variable in terms of flow-sensitivity
- Pre-trained by an off-the-shelf method*

***Learning a Strategy for Adapting a Program Analysis via Bayesian Optimisation, OOPSLA'15**

Model

- Model *M :* Variable → [0, 1]
- Importance of each variable in terms of flow-sensitivity
- Pre-trained by an off-the-shelf method*

```
x = 0; y = 0; z = 1; v = input(); w = input();
x = z;
z = z + 1;
y = x;
assert(y > 0);    // Query 1 (hold)
assert(z > 0);    // Query 2 (hold)
assert(v == w);   // Query 3 (may fail)
```

***Learning a Strategy for Adapting a Program Analysis via Bayesian Optimisation, OOPSLA'15**

Model

- Model *M :* Variable → [0, 1]
- Importance of each variable in terms of flow-sensitivity
- Pre-trained by an off-the-shelf method*

```
x = 0; y = 0; z = 1; v = input(); w = input();
x = z;
z = z + 1;
y = x;
assert(y > 0);    // Query 1 (hold)
assert(z > 0);    // Query 2 (hold)
assert(v == w);   // Query 3 (may fail)
```

$> M(w)$

***Learning a Strategy for Adapting a Program Analysis via Bayesian Optimisation, OOPSLA'15**

Model

- Model *M :* Variable → [0, 1]
- Importance of each variable in terms of flow-sensitivity
- Pre-trained by an off-the-shelf method*

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);   // Query 3 (may fail)
```

M(x) > ***> M(w)***

*Learning a Strategy for Adapting a Program Analysis via Bayesian Optimisation, OOPSLA'15

Model

- Model *M* : Variable → [0, 1]
- Importance of each variable in terms of flow-sensitivity
- Pre-trained by an off-the-shelf method*

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);   // Query 3 (may fail)
```

$$M(x) > M(y) > M(z) > M(v) > M(w)$$

*Learning a Strategy for Adapting a Program Analysis via Bayesian Optimisation, OOPSLA'15

Controller

Controller

- Controller $\boldsymbol{\pi} : \mathbf{F} \rightarrow \mathrm{Pr}(\mathrm{A})$ where $\mathbf{A} = \{\mathbf{0}, \cdots, \mathbf{100}\}$

Controller

- Controller $\boldsymbol{\pi} : \mathbf{F} \rightarrow \Pr(\mathrm{A})$ where $\mathrm{A} = \{0, \cdots, 100\}$
- Input: a feature vector describing current status
  - e.g., memory usage, analysis progress, etc

Controller

- Controller $\boldsymbol{\pi} : \mathbf{F} \rightarrow \Pr(\mathrm{A})$ where $\mathrm{A} = \{0, \cdots, 100\}$
- Input: a feature vector describing current status
  - e.g., memory usage, analysis progress, etc
- Output: probability distribution on % of variables that should be treated flow-insensitively

Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
x = 0; y = 0; z = 1; v = input(); w = input();
x = z;
z = z + 1;
y = x;
assert(y > 0);     // Query 1 (hold)
assert(z > 0);     // Query 2 (hold)
assert(v == w);    // Query 3 (may fail)
```

Model: $M(x) > M(y) > M(z) > M(v) > M(w)$

Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);   // Query 3 (may fail)
```

Model: $M(x) > M(y) > M(z) > M(v) > M(w)$

Line	Flow-Sensitive Abstract State
1	{x = [0,0], y = [0,0], z = [1,1], v = ⊤, w = ⊤}
2	{x = [1,1], y = [0,0], z = [1,1], v = ⊤, w = ⊤}

6 Intervals

Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);   // Query 3 (may fail)
```

Model: $M(x) > M(y) > M(z) > M(v) \cancel{> M(w)}$

Line	Flow-Sensitive	Flow-Insensitive
1	{x = [0,0], y = [0,0], z = [1,1], v = ⊤}	{w = ⊤}
2	{x = [1,1], y = [0,0], z = [1,1], v = ⊤}	

6 Intervals

Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);     // Query 1 (hold)
6: assert(z > 0);     // Query 2 (hold)
7: assert(v == w);    // Query 3 (may fail)
```

Model: $M(x) > M(y) > M(z) > M(v) \cancel{> M(w)}$

Line	Flow-Sensitive	Flow-Insensitive
1	{x = [0,0], y = [0,0], z = [1,1], v = ⊤}	
2	{x = [1,1], y = [0,0], z = [1,1], v = ⊤}	{w = ⊤}
3	{x = [1,1], y = [0,0], z = [2,2], v = ⊤}	

9 Intervals

Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);     // Query 1 (hold)
6: assert(z > 0);     // Query 2 (hold)
7: assert(v == w);    // Query 3 (may fail)
```

Model: $M(x) > M(y) \cancel{> M(z) > M(v) > M(w)}$

Line	Flow-Sensitive	Flow-Insensitive
1	{x = [0,0], y = [0,0]}	{z = [1,+∞], v = ⊤, w = ⊤}
2	{x = [1,+∞], y = [0,0]}	
3	{x = [1,+∞], y = [0,0]}	

6 Intervals

Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);     // Query 1 (hold)
6: assert(z > 0);     // Query 2 (hold)
7: assert(v == w);    // Query 3 (may fail)
```

Model: $M(x) > M(y)$ ~~$> M(z) > M(v) > M(w)$~~

Line	Flow-Sensitive	Flow-Insensitive
1	{x = [0,0], y = [0,0]}	{z = [1,+∞], v = ⊤, w = ⊤}
2	{x = [1,+∞], y = [0,0]}	
3	{x = [1,+∞], y = [0,0]}	
4	**{x = [1,+∞], y = [1,+∞]}**	

8 Intervals

Learning Controller

- Controller $\boldsymbol{\pi} : \mathbf{F} \rightarrow \mathbf{Pr(A)}$ where $\mathbf{A} = \{0, \cdots, 100\}$
  - Input: a feature vector describing the current status
  - Output: probability distribution on % of variables that should be treated flow-insensitively

Learning Controller

- Controller $\boldsymbol{\pi} : \mathbf{F} \rightarrow \mathrm{Pr}(\mathrm{A})$ where $\mathrm{A} = \{0, \cdots, 100\}$
  - Input: a feature vector describing the current status
  - Output: probability distribution on % of variables that should be treated flow-insensitively

- Value function $Q : \mathbf{F} \times \mathbf{A} \rightarrow [0, 1]$
  - Score to every pair of feature vector and action

Learning Controller

- Controller $\boldsymbol{\pi} : \mathbf{F} \to \mathbf{Pr(A)}$ where $\mathbf{A} = \{0, \cdots, 100\}$
  - Input: a feature vector describing the current status
  - Output: probability distribution on % of variables that should be treated flow-insensitively

- Value function $Q : \mathbf{F} \times \mathbf{A} \to [0, 1]$
  - Score to every pair of feature vector and action

- $\boldsymbol{\pi}_Q(\mathrm{f})(\mathrm{a}) = \dfrac{Q(f,a)}{\sum_{a' \in A} Q(f,a')}$

Value Function

$$Q : \mathbf{F} \times \mathbf{A} \rightarrow [0, 1]$$

Value Function

$$Q : \mathbf{F} \times \mathbf{A} \rightarrow [0, 1]$$

- Feature abstraction function α : State $\rightarrow \mathbf{F}$ where $\mathbf{F} = [0, 1]^4$

1. The inverse of memory budget
2. Current memory consumption divided by the total budget
3. Current lattice position divided by the lattice height
4. Current workset size divided by the total workset size

Value Function

$$Q : \mathbf{F} \times \mathbf{A} \rightarrow [0, 1]$$

- Feature abstraction function $\boldsymbol{\alpha}$: State $\rightarrow \mathbf{F}$ where $\mathbf{F} = [0, 1]^4$

  1. The inverse of memory budget
  2. Current memory consumption divided by the total budget
  3. Current lattice position divided by the lattice height
  4. Current workset size divided by the total workset size

- Reward : [0, 1]
  - relative #alarms w.r.t. flow-sensitive and insensitive result
    - 0 if #alarms == #flow-insensitive alarms
    - 1 if #alarms == #flow-sensitive alarms

Learning Algorithm

Learning Algorithm

- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)

Learning Algorithm

- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)

$\boxed{\textbf{1. Initialize } \pi \textbf{ with a random policy}}$

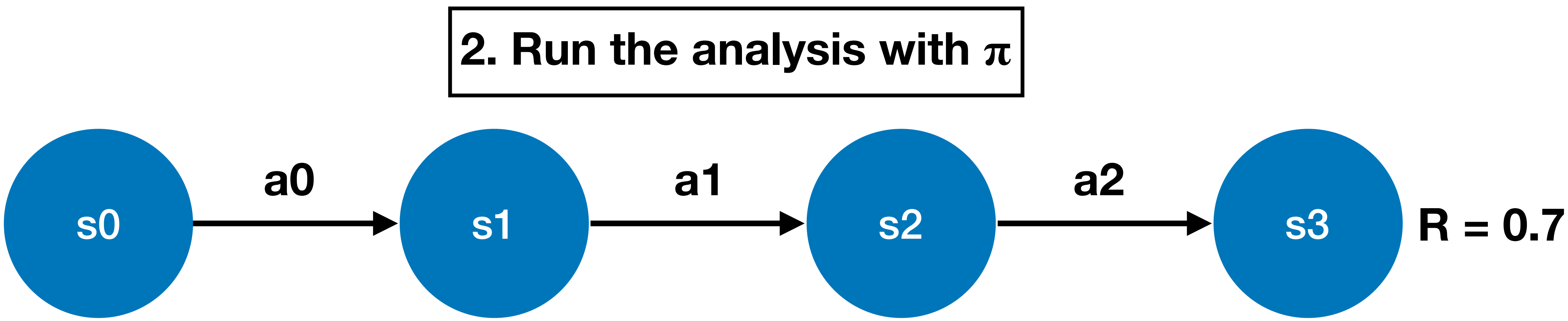
Learning Algorithm

- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)

$\boxed{\textbf{2. Run the analysis with } \pi}$

Learning Algorithm

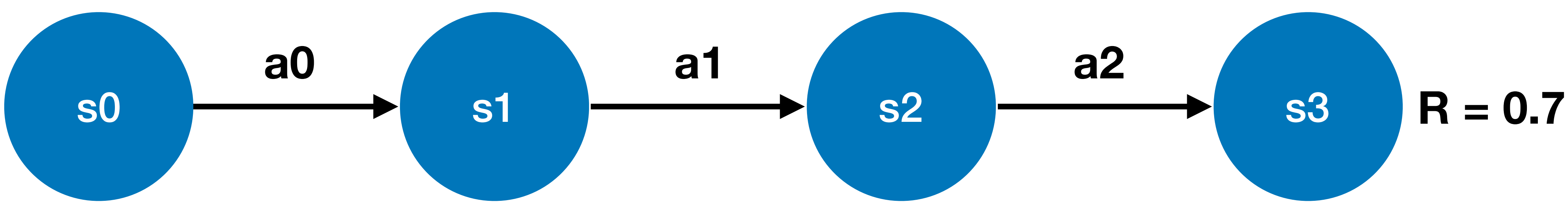
- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)



Learning Algorithm

- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)

3. Collect all state-action pairs and the reward



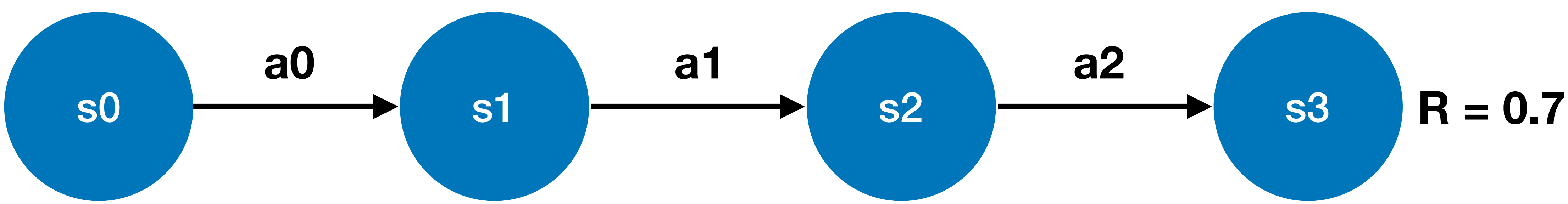
$$D_1 = \{(<\alpha(s_0), a_0>, 0.7), (<\alpha(s_1), a_1>, 0.7), (<\alpha(s_2), a_2>, 0.7)\}$$

*For brevity heuristics are omitted

Learning Algorithm

- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)

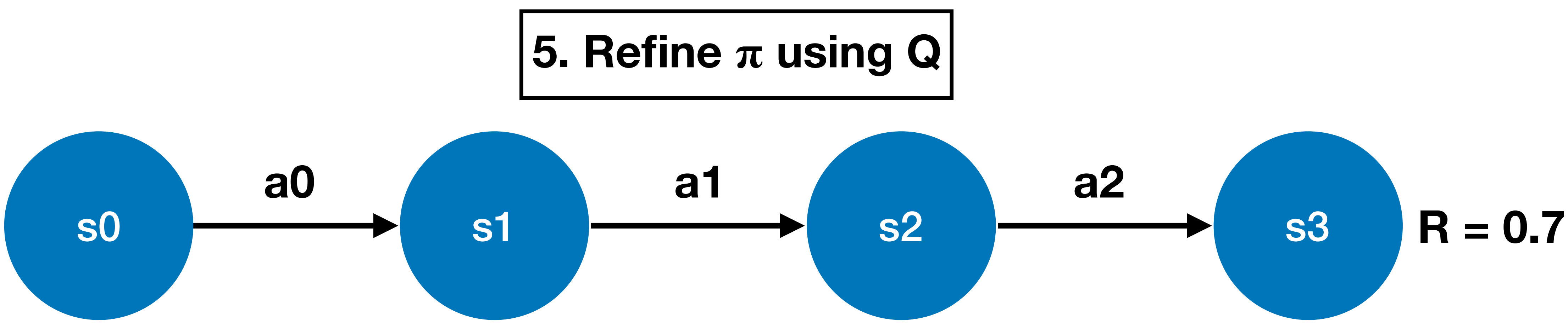
4. Learn Q using D_1 with a supervised learning algorithm



Q = SupervisedLearning(D_1)

Learning Algorithm

- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)

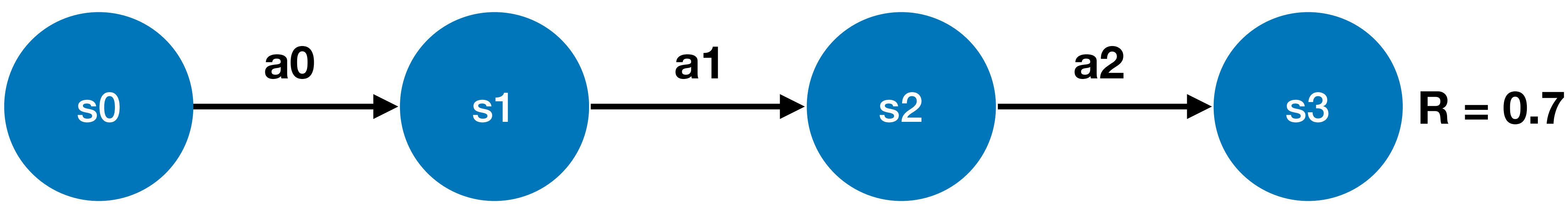


$$\pi_Q(f)(a) = \frac{Q(f,a)}{\sum_{a' \in A} Q(f,a')}$$

Learning Algorithm

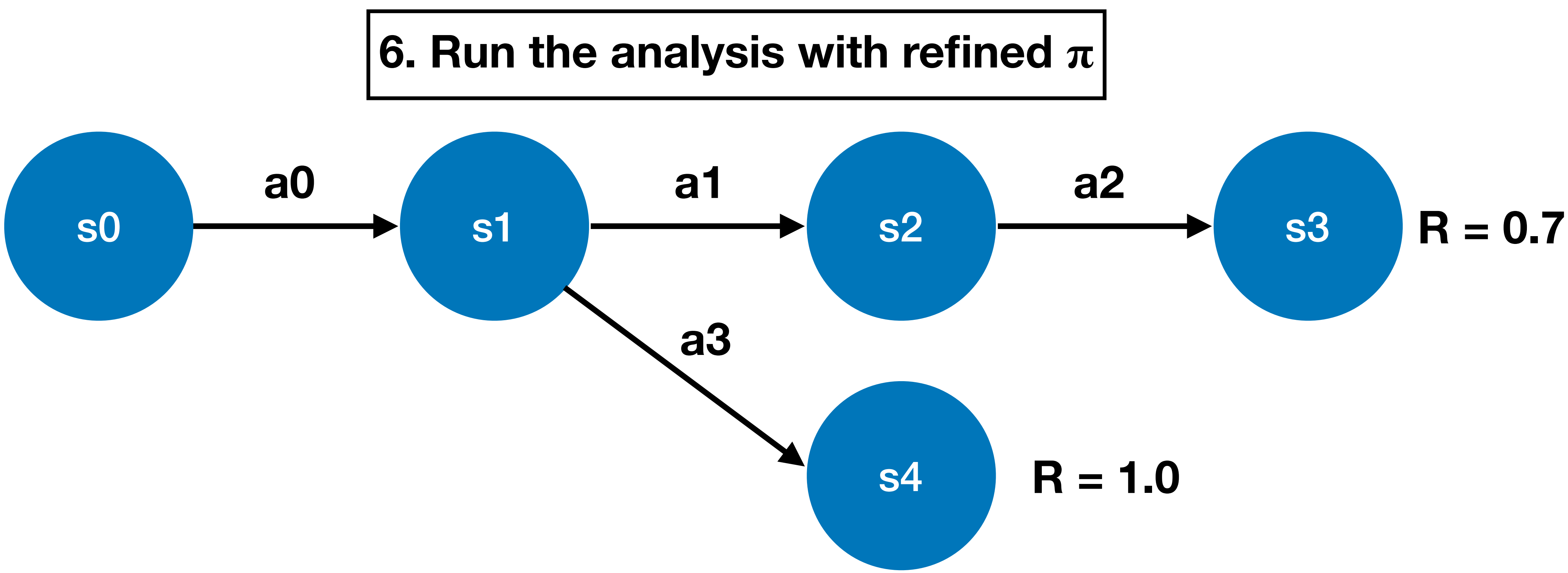
- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)

6. Run the analysis with refined π



Learning Algorithm

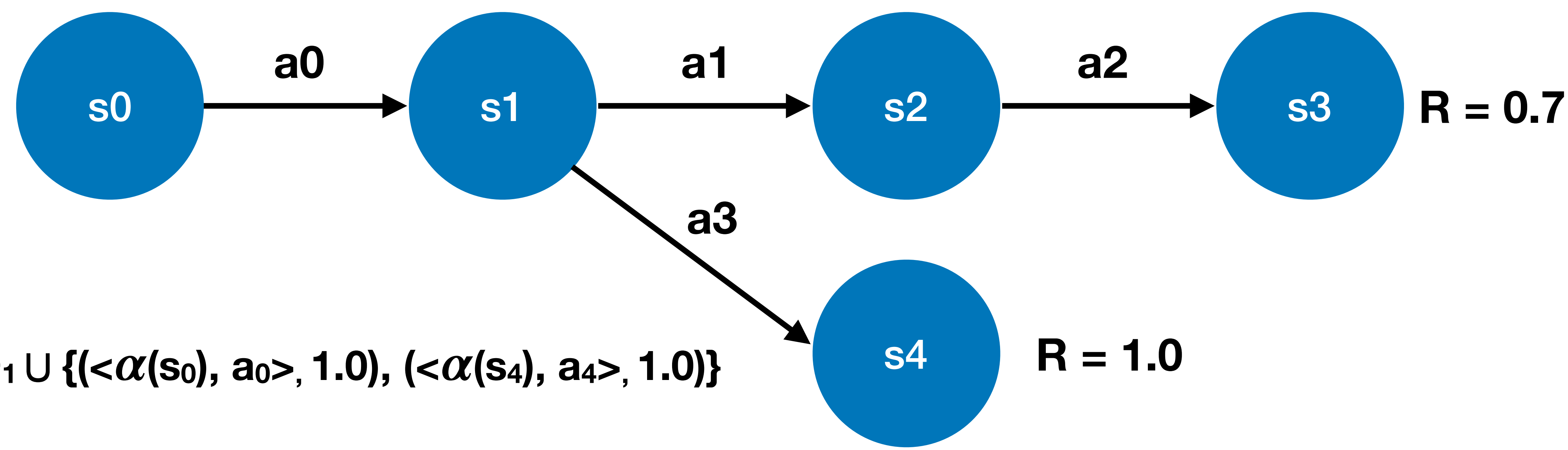
- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)



Learning Algorithm

- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)

7. Accumulate data

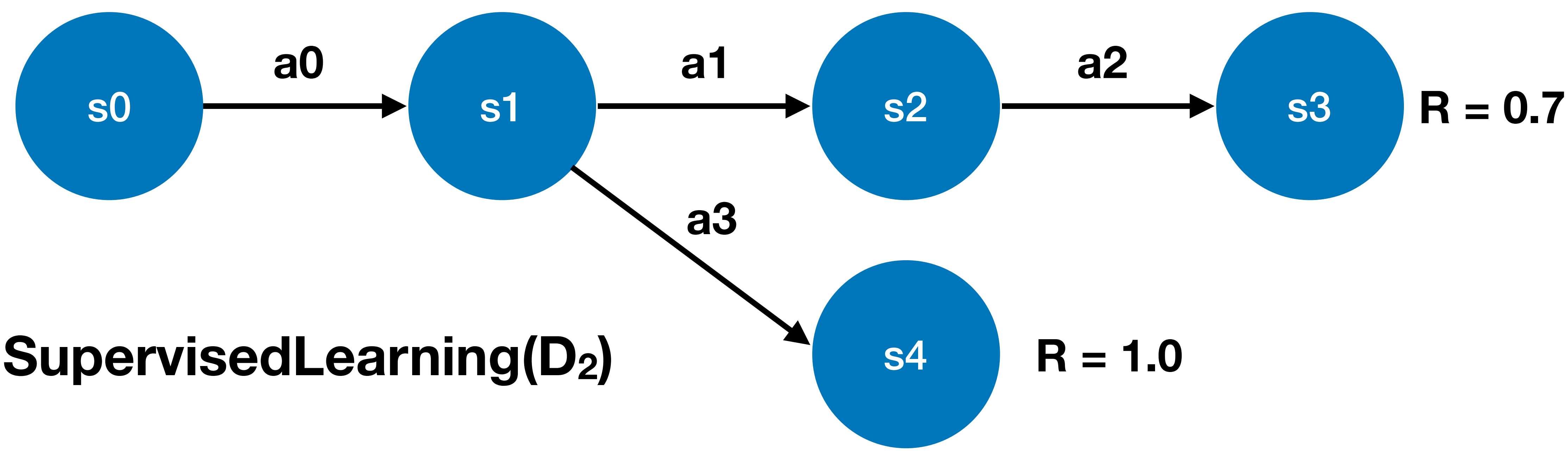


$$D_2 = D_1 \cup \{(<\alpha(s_0), a_0>, 1.0), (<\alpha(s_4), a_4>, 1.0)\}$$

Learning Algorithm

- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)

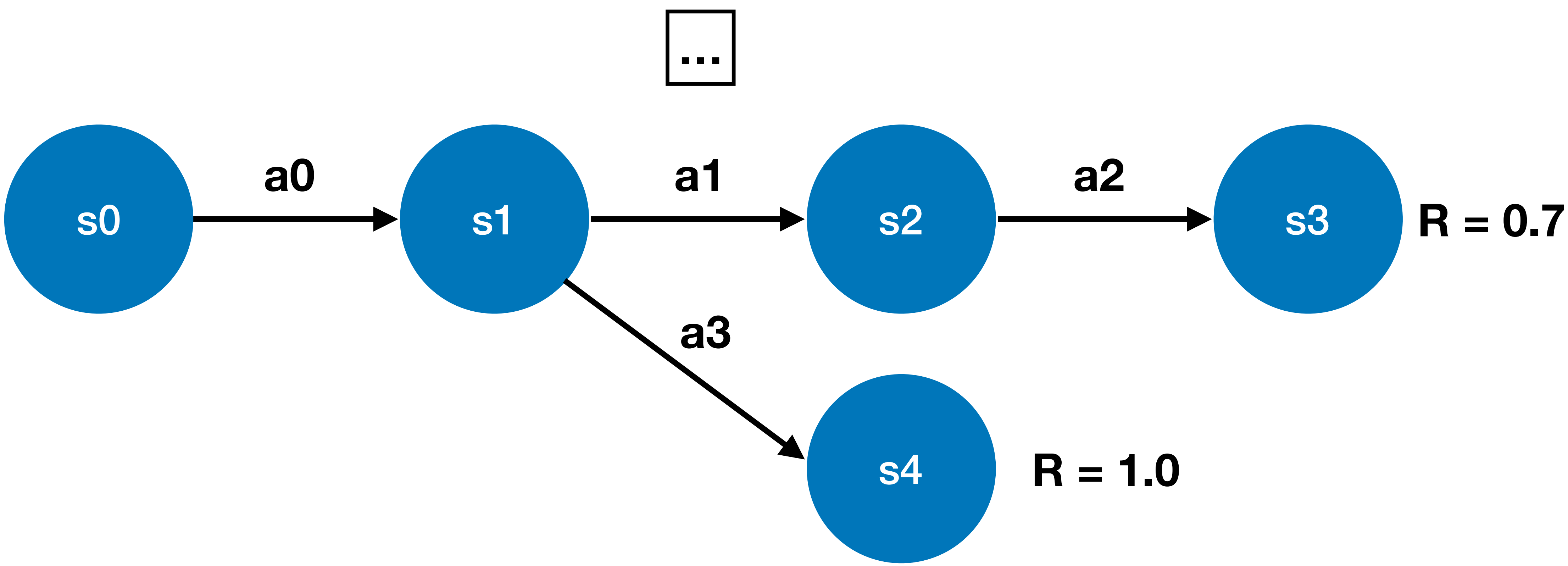
8. Refine Q using D_2 with a supervised learning algorithm



Q = SupervisedLearning(D_2)

Learning Algorithm

- SARSA-style algorithm from reinforcement learning
  - from a training set (i.e., batch mode)
  - with common heuristics (discounted reward, e-greedy search)



Outline

- Motivation
- Learning Framework
- **Experimental Results**
- Conclusion

Experimental Setup

Experimental Setup

- Training with 10 small programs (15—80KLOC)
  - with small memory limits

Experimental Setup

- Training with 10 small programs (15—80KLOC)
  - with small memory limits
- Test with 8 large programs (129—503KLOC)
  - 64 / 128 GB memory limits

Experimental Setup

- Training with 10 small programs (15—80KLOC)
    - with small memory limits
- Test with 8 large programs (129—503KLOC)
    - 64 / 128 GB memory limits
- Measure buffer-overrun and null-dereference alarms

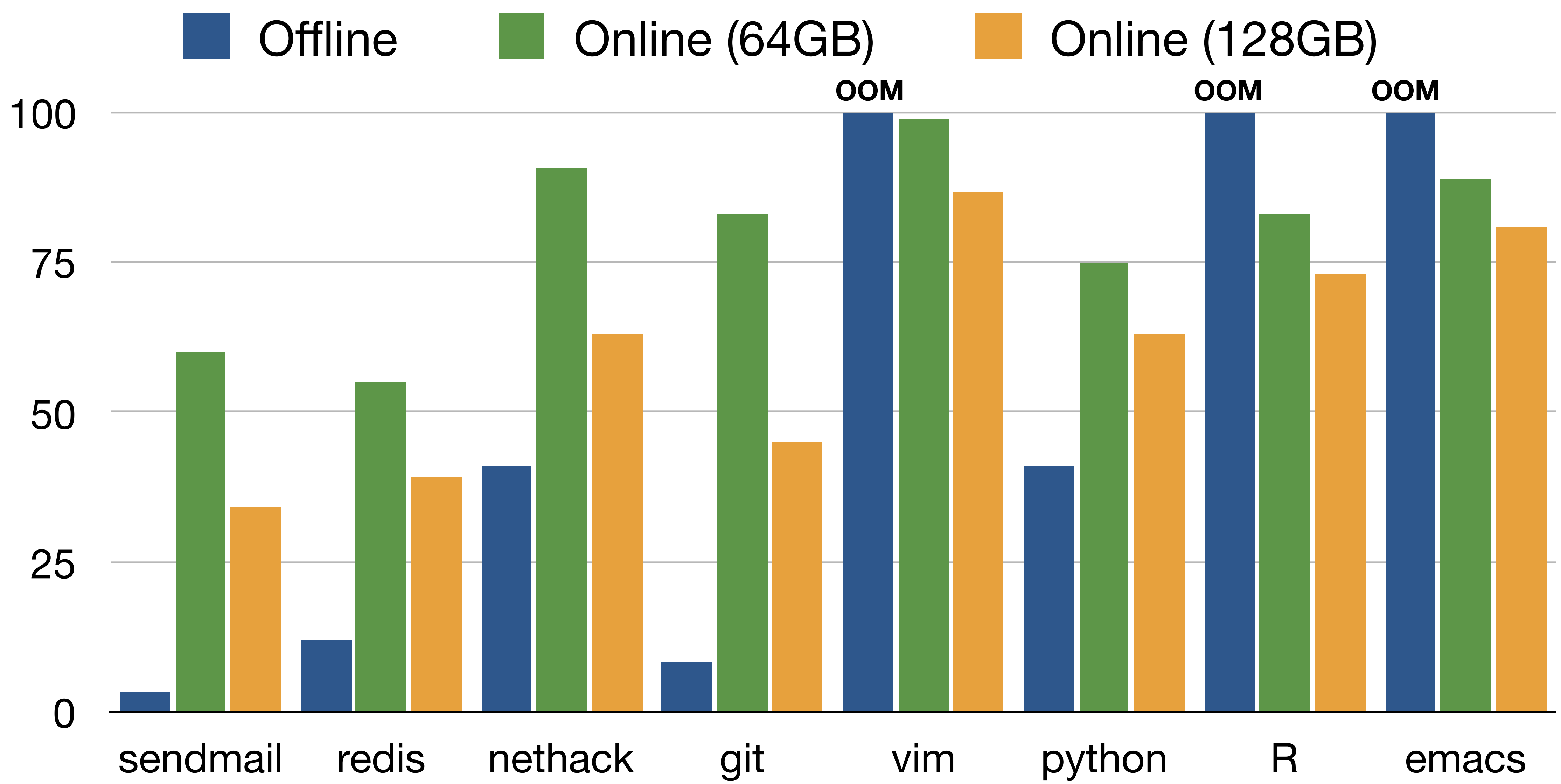
Experimental Setup

- Training with 10 small programs (15—80KLOC)
  - with small memory limits
- Test with 8 large programs (129—503KLOC)
  - 64 / 128 GB memory limits
- Measure buffer-overrun and null-dereference alarms
- Trigger controller when the OCaml runtime allocates new memory chunks

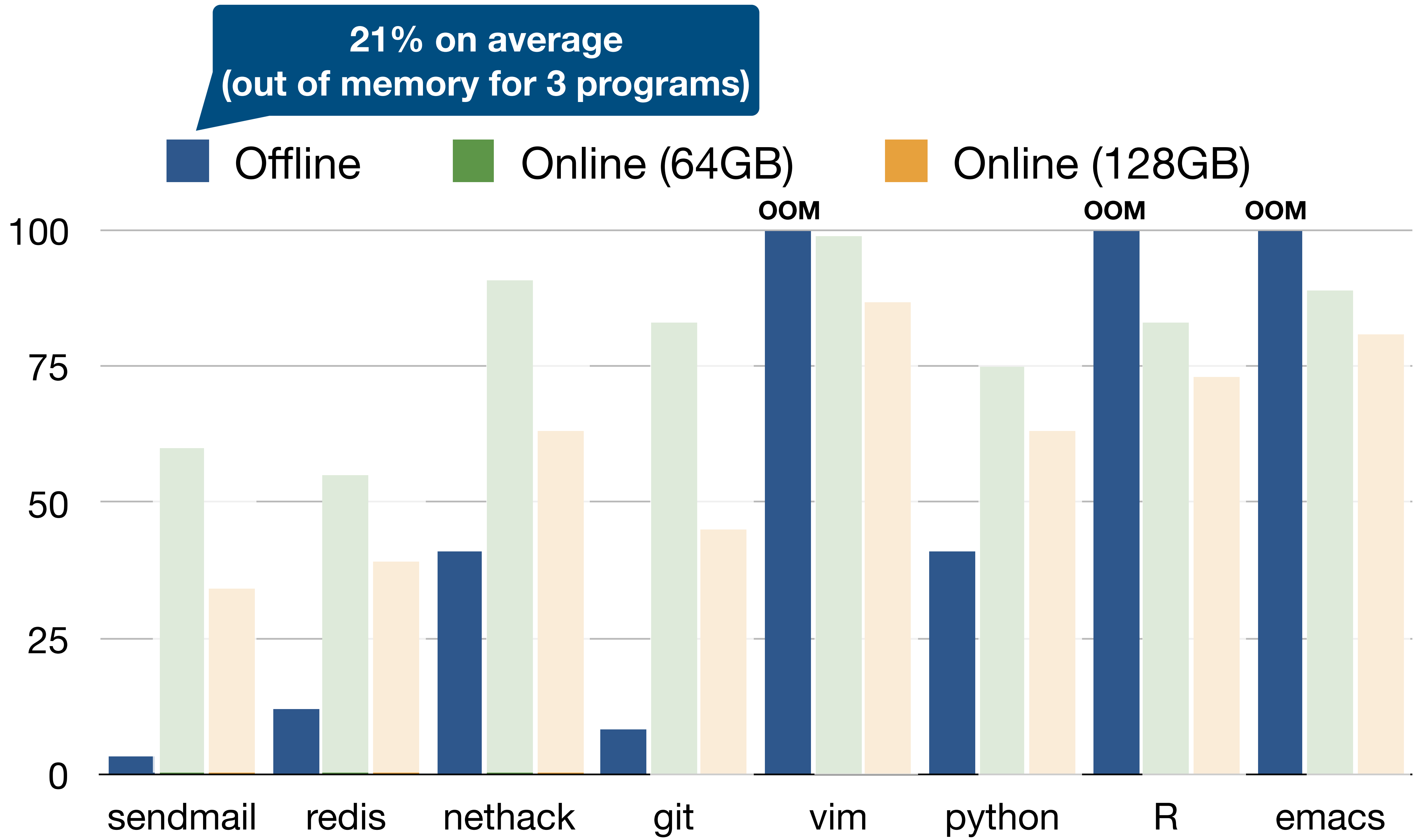
Experimental Setup

- Training with 10 small programs (15—80KLOC)
  - with small memory limits
- Test with 8 large programs (129—503KLOC)
  - 64 / 128 GB memory limits
- Measure buffer-overrun and null-dereference alarms
- Trigger controller when the OCaml runtime allocates new memory chunks
- Compared to partially flow-sensitive analysis
  - 10% of variables chosen offline with 128GB of memory

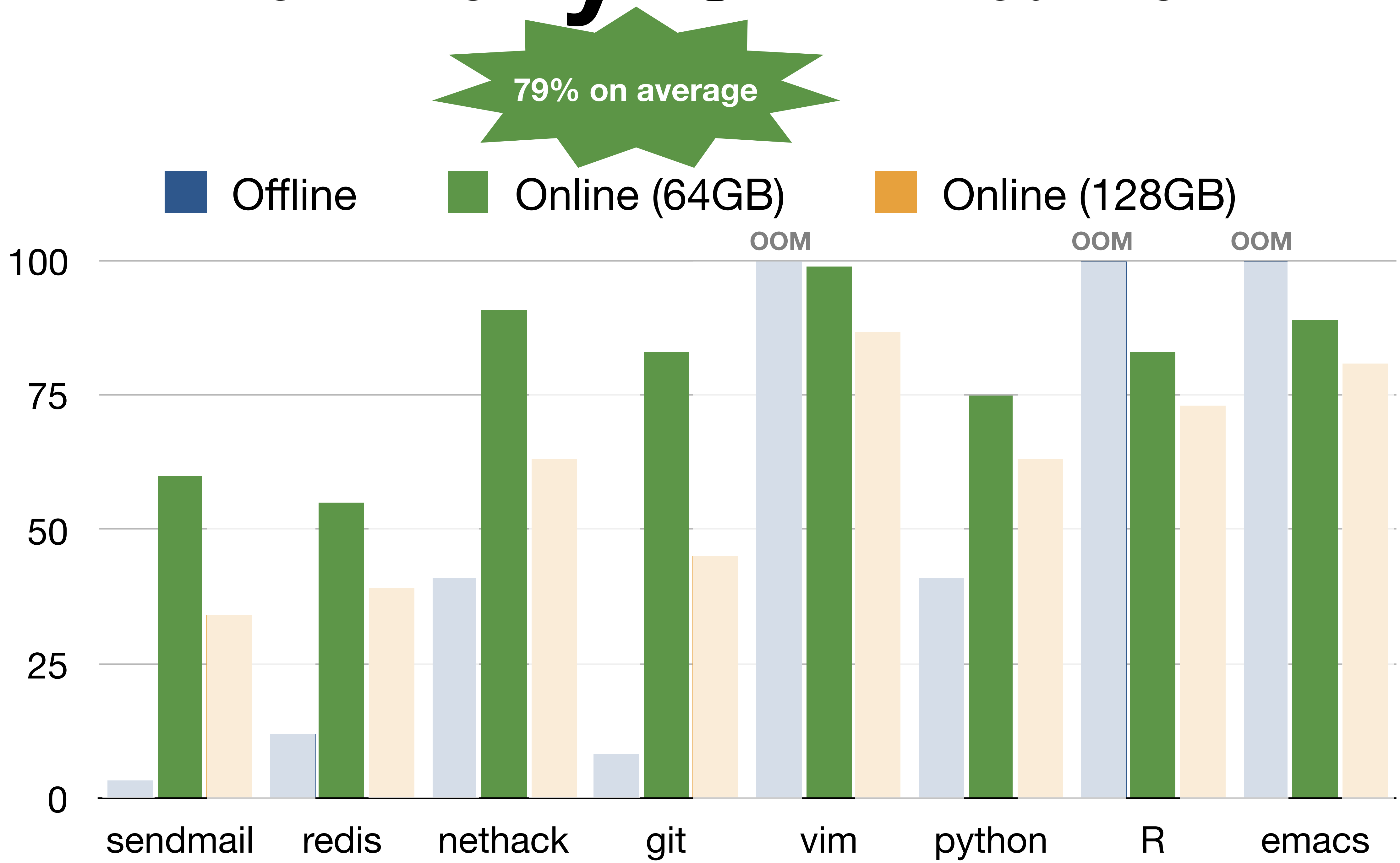
Memory Utilization



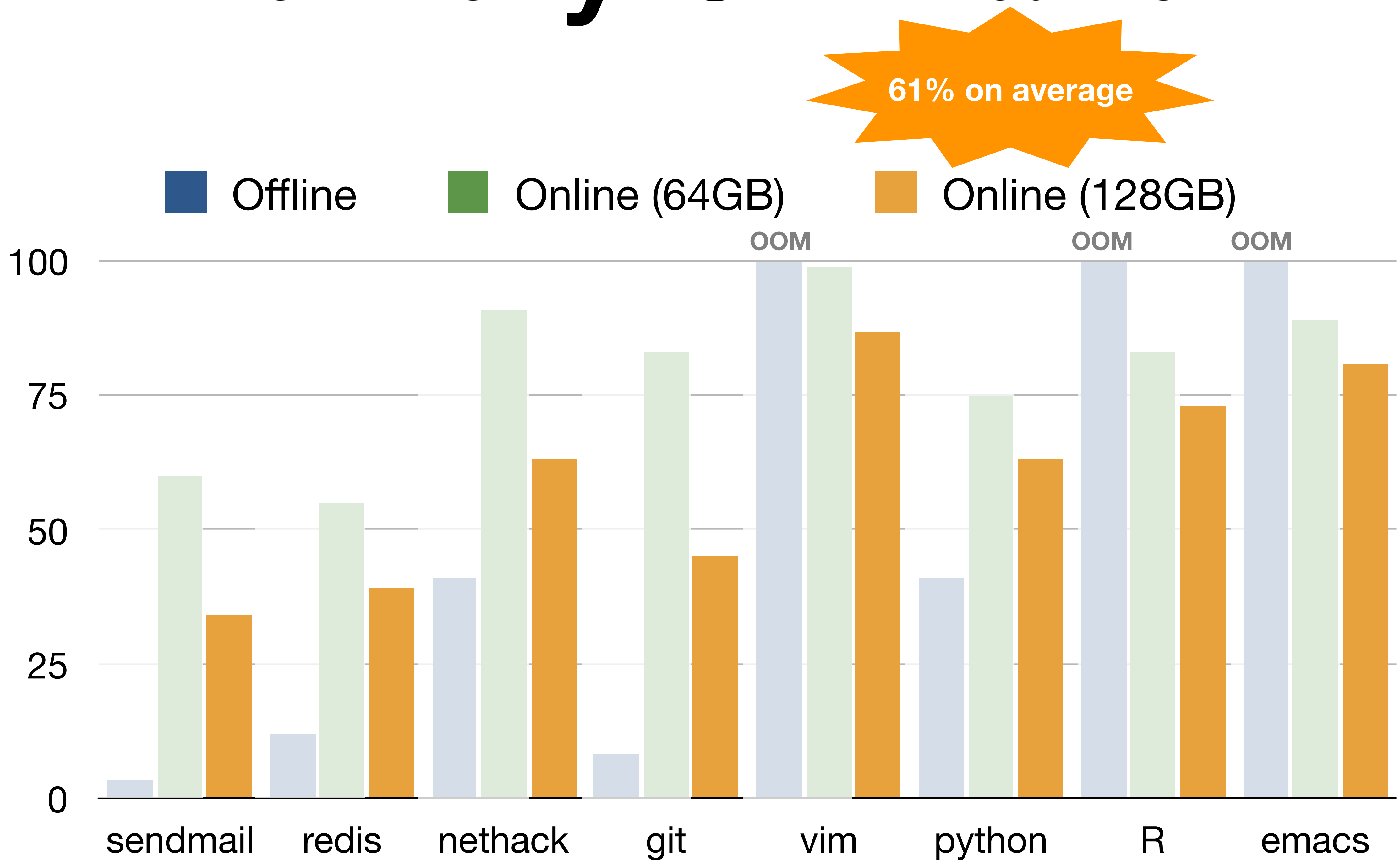
Memory Utilization



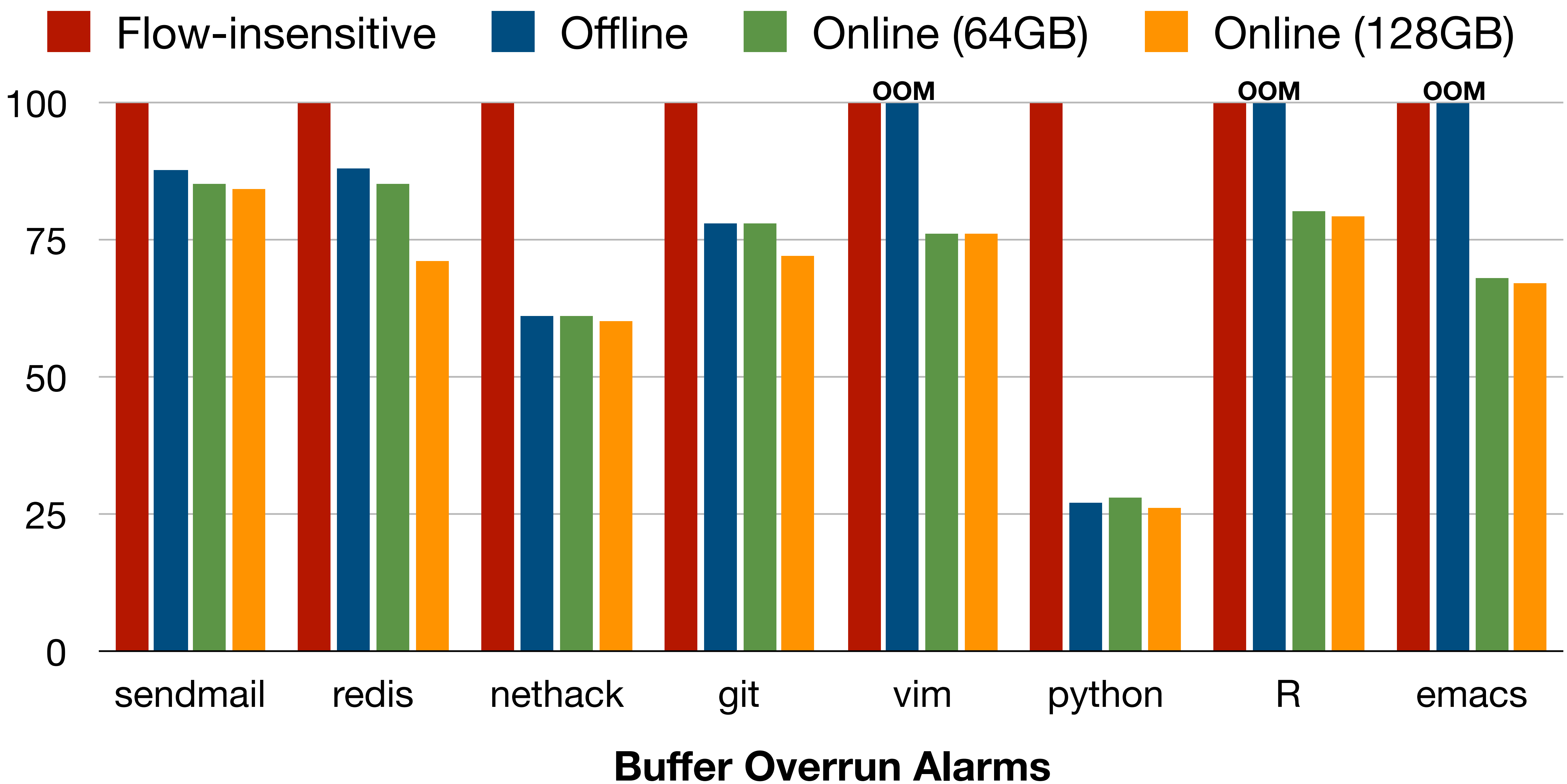
Memory Utilization



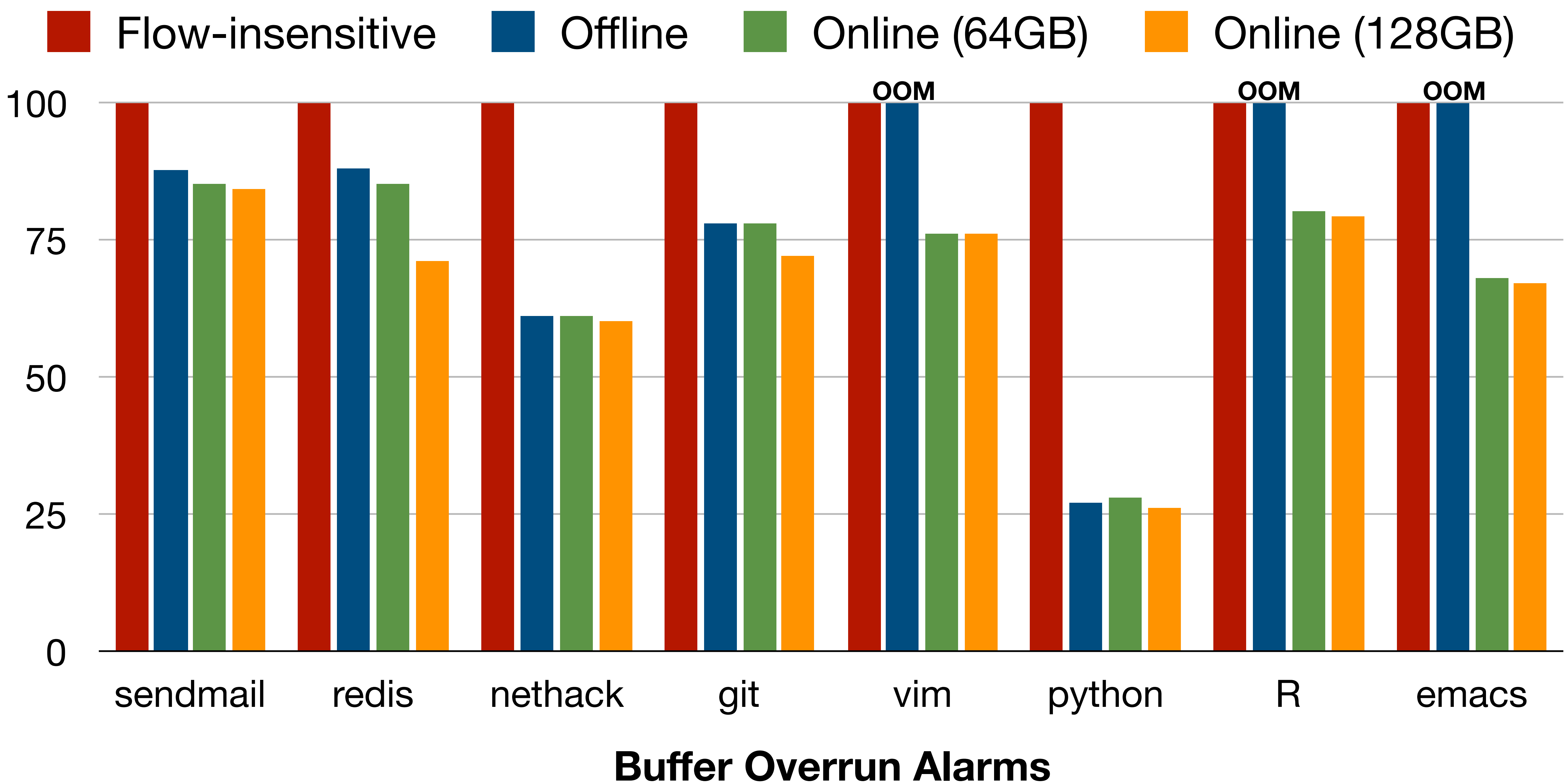
Memory Utilization



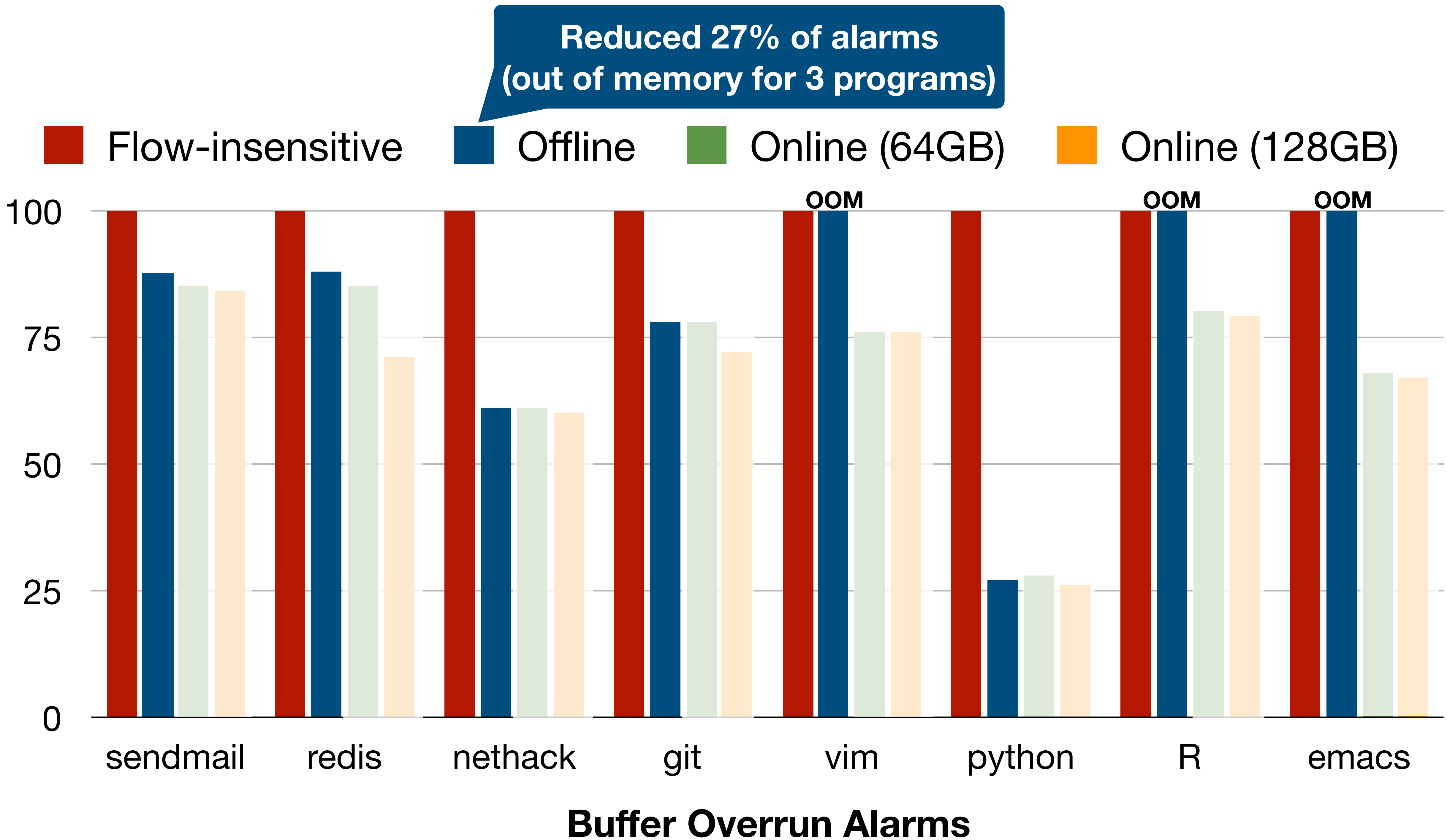
Analysis Precision



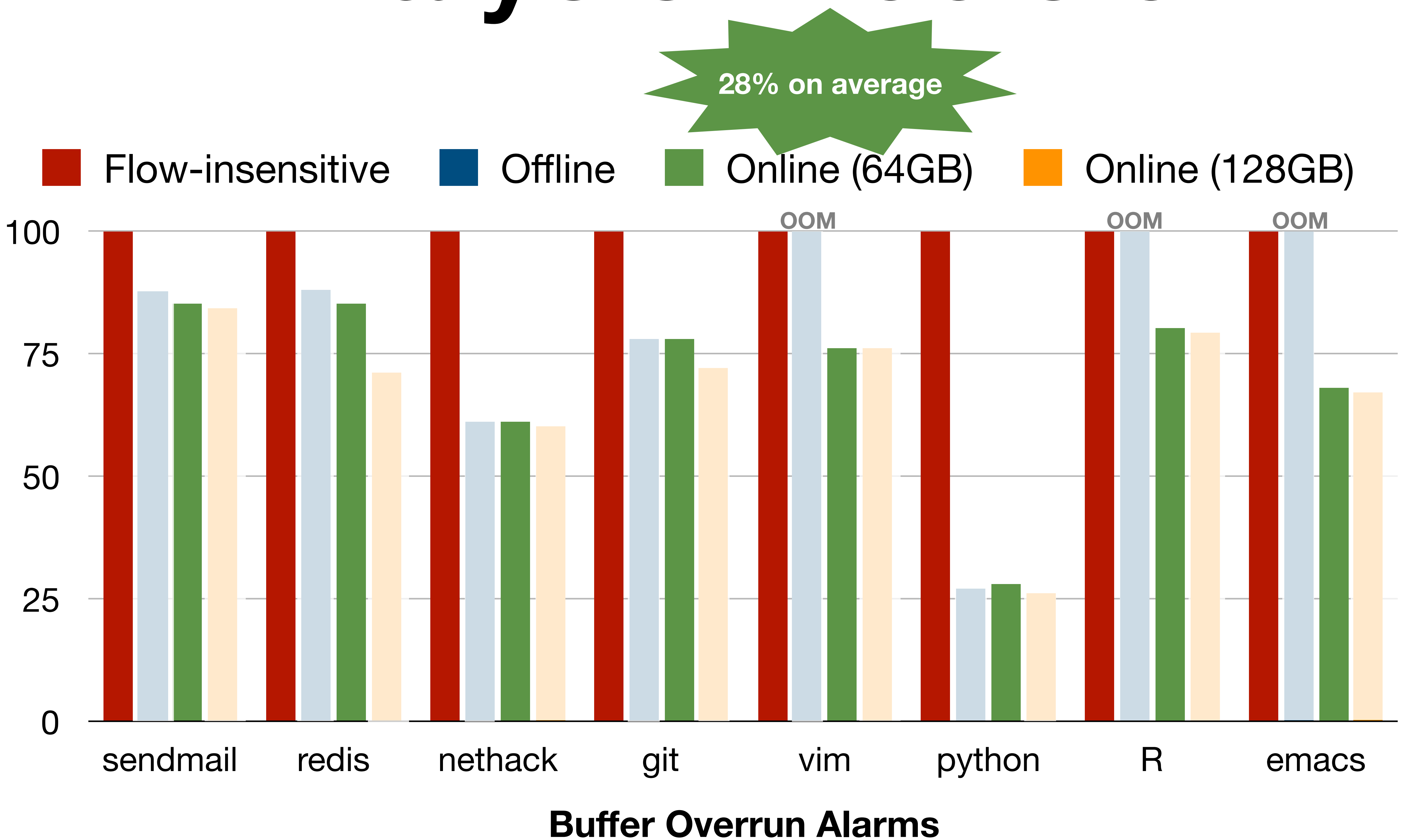
Analysis Precision



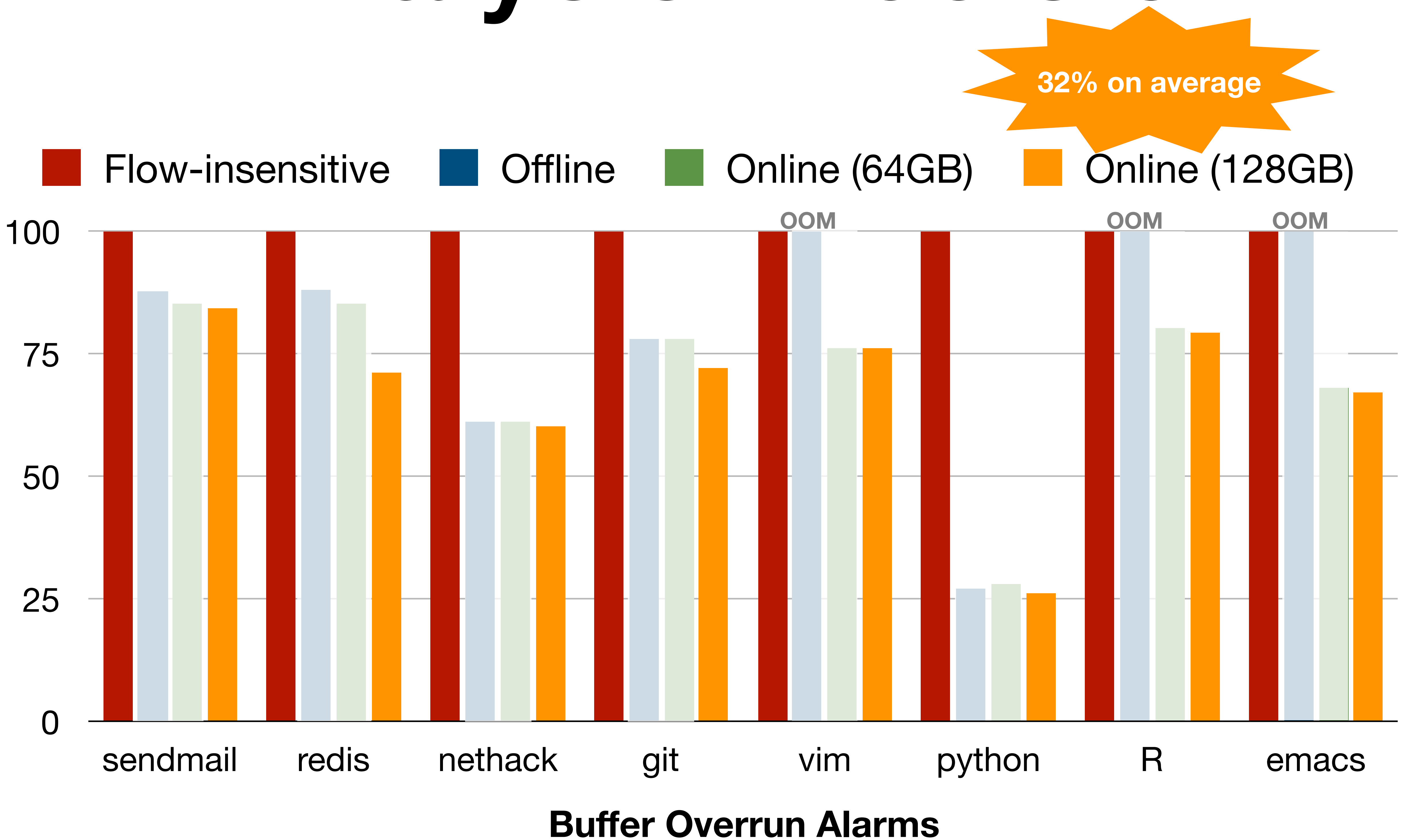
Analysis Precision



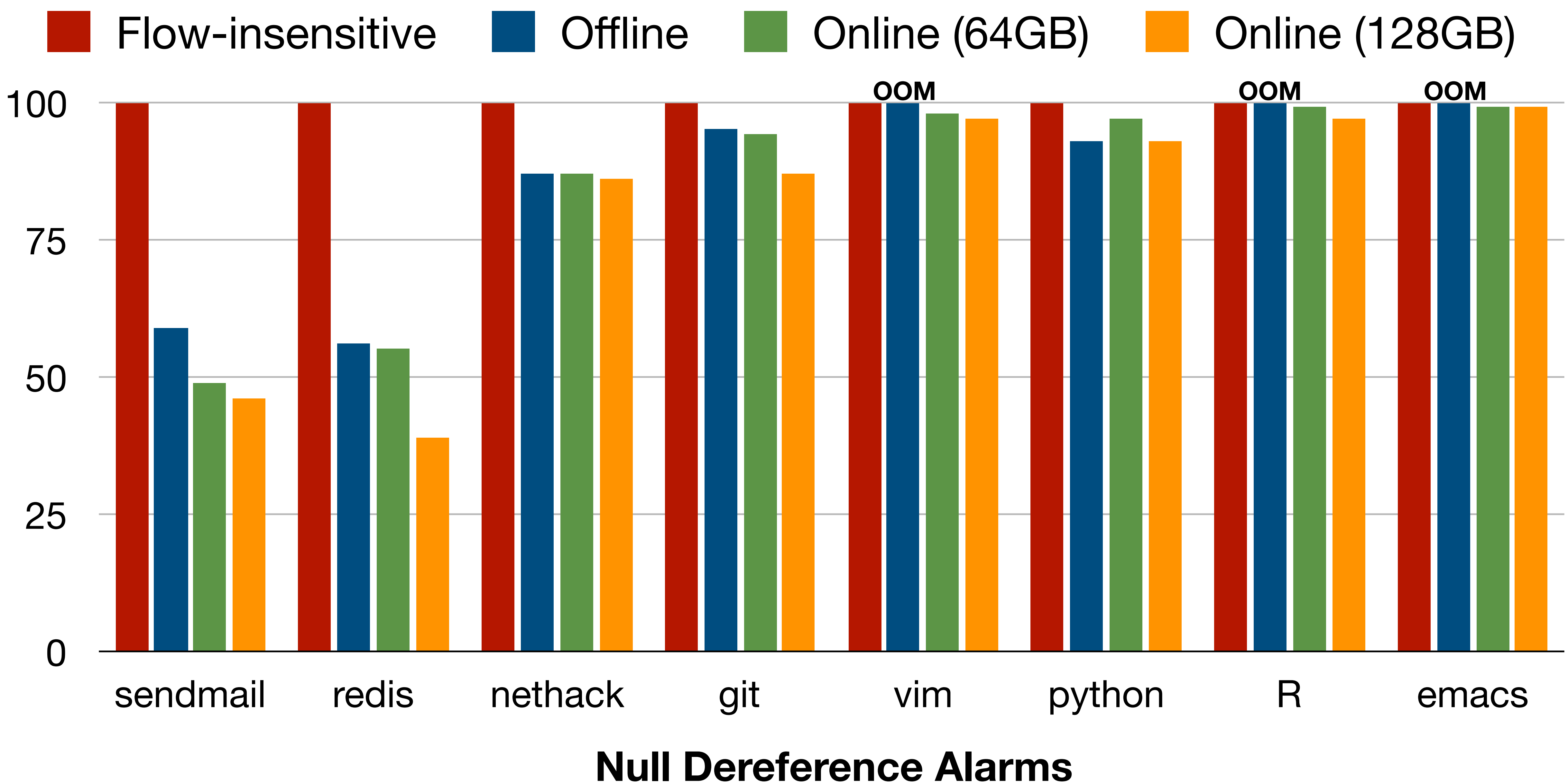
Analysis Precision



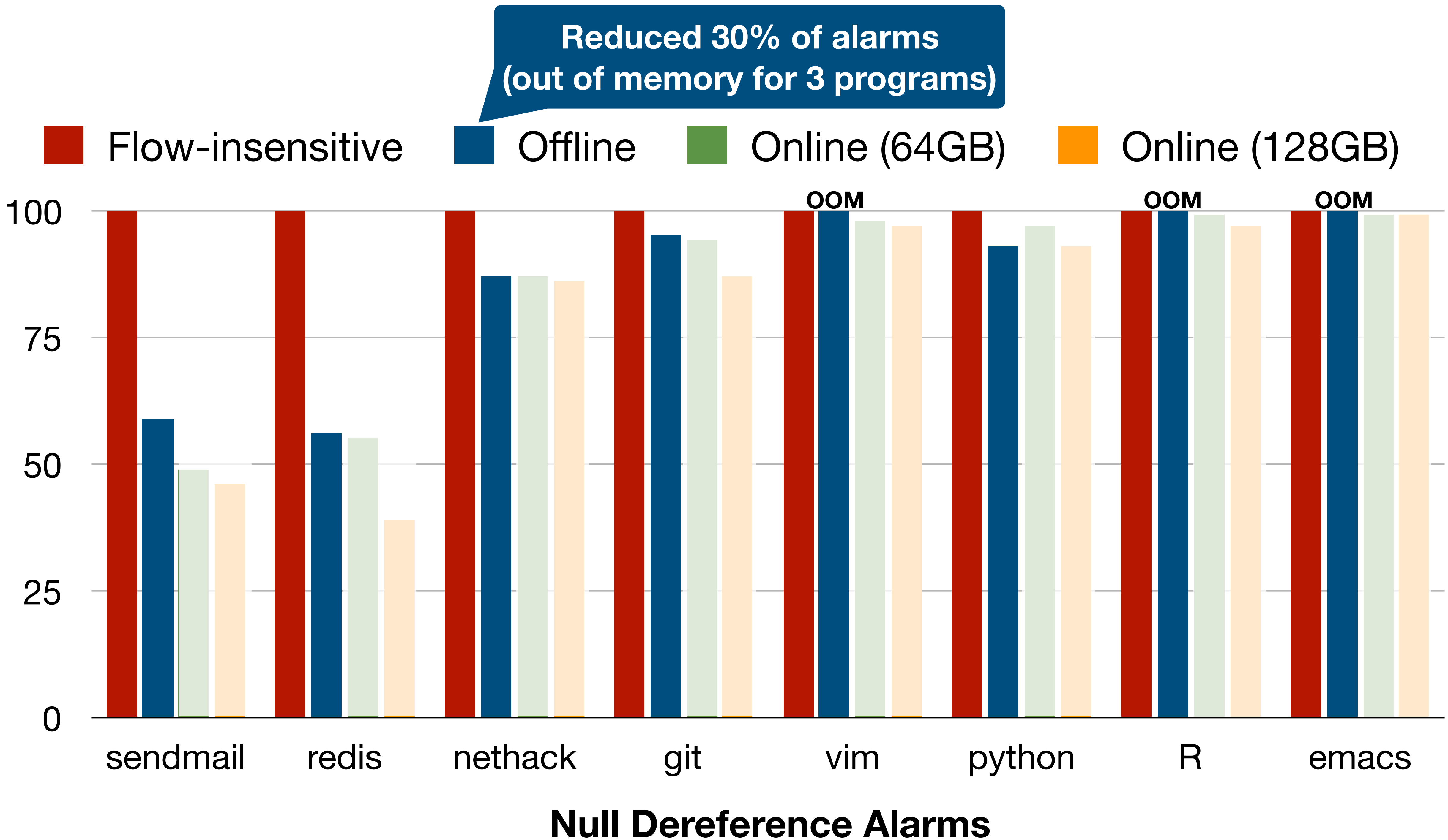
Analysis Precision



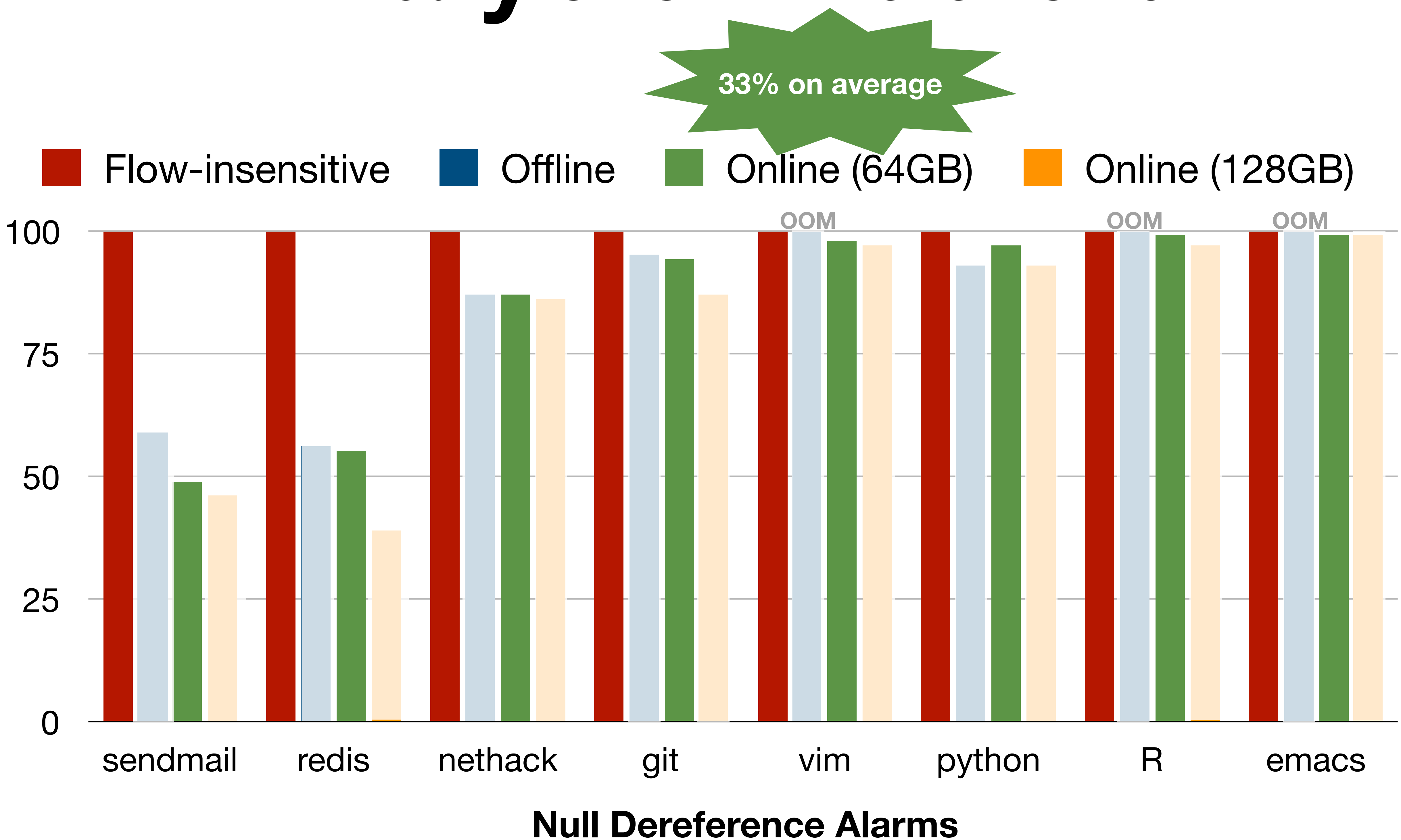
Analysis Precision



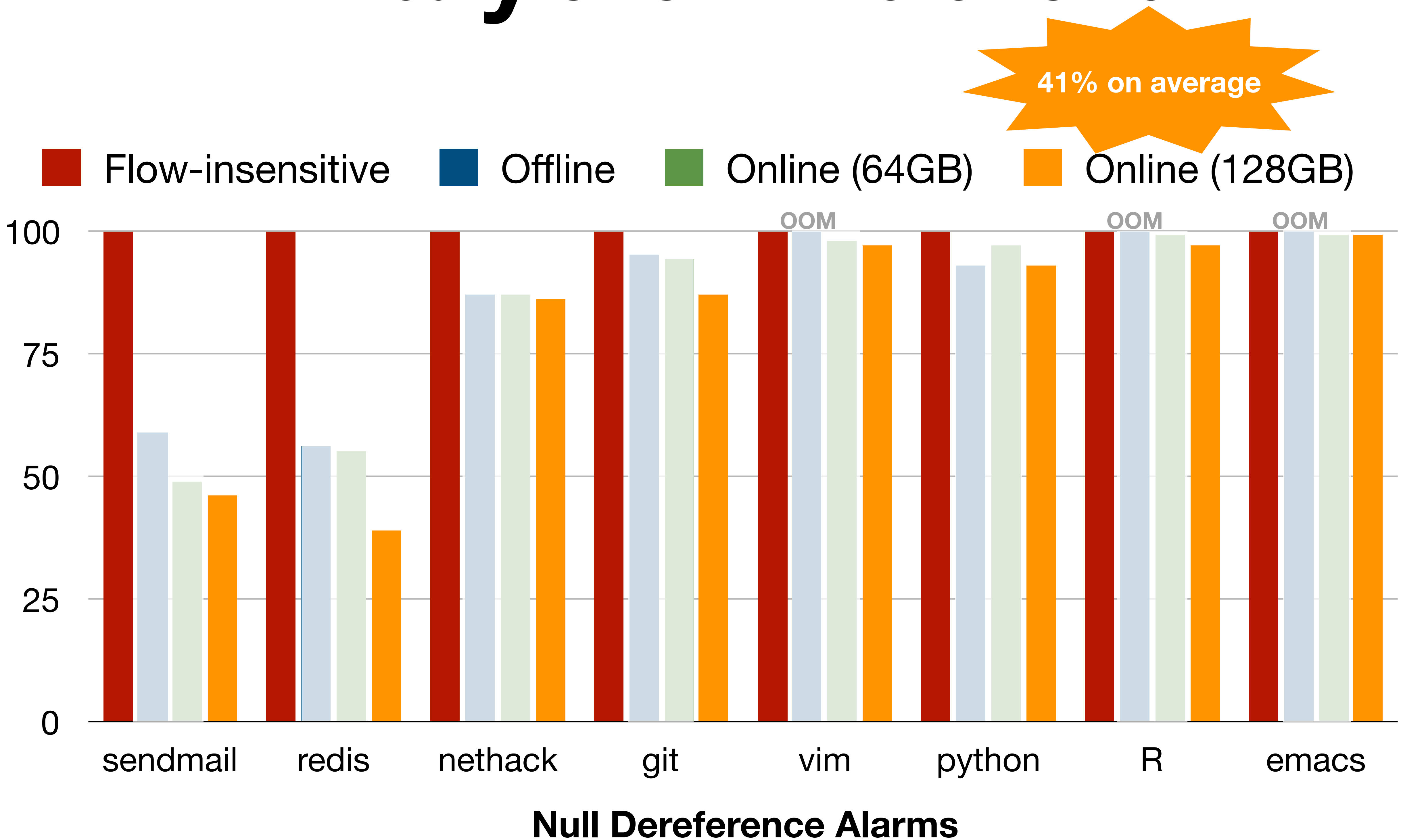
Analysis Precision



Analysis Precision



Analysis Precision



Conclusion

Conclusion

- A systematic framework for resource-aware program analysis
  - **online** abstraction coarsening
  - **reinforcement learning** algorithm for learning controller
  - attention to **physical resource** as well as logical behavior

Conclusion

- A systematic framework for resource-aware program analysis
  - **online** abstraction coarsening
  - **reinforcement learning** algorithm for learning controller
  - attention to **physical resource** as well as logical behavior