コンパイラ演習
第 9 回
(2011/12/08)

中村 晃一　野瀬 貴史　前田 俊行
秋山 茂樹　池尻 拓朗
鈴木 友博　渡邊 裕貴
潮田 資秀
小酒井 隆広
山下 諒蔵　佐藤 春旗
大山 恵弘　佐藤 秀明
住井 英二郎

今回の内容

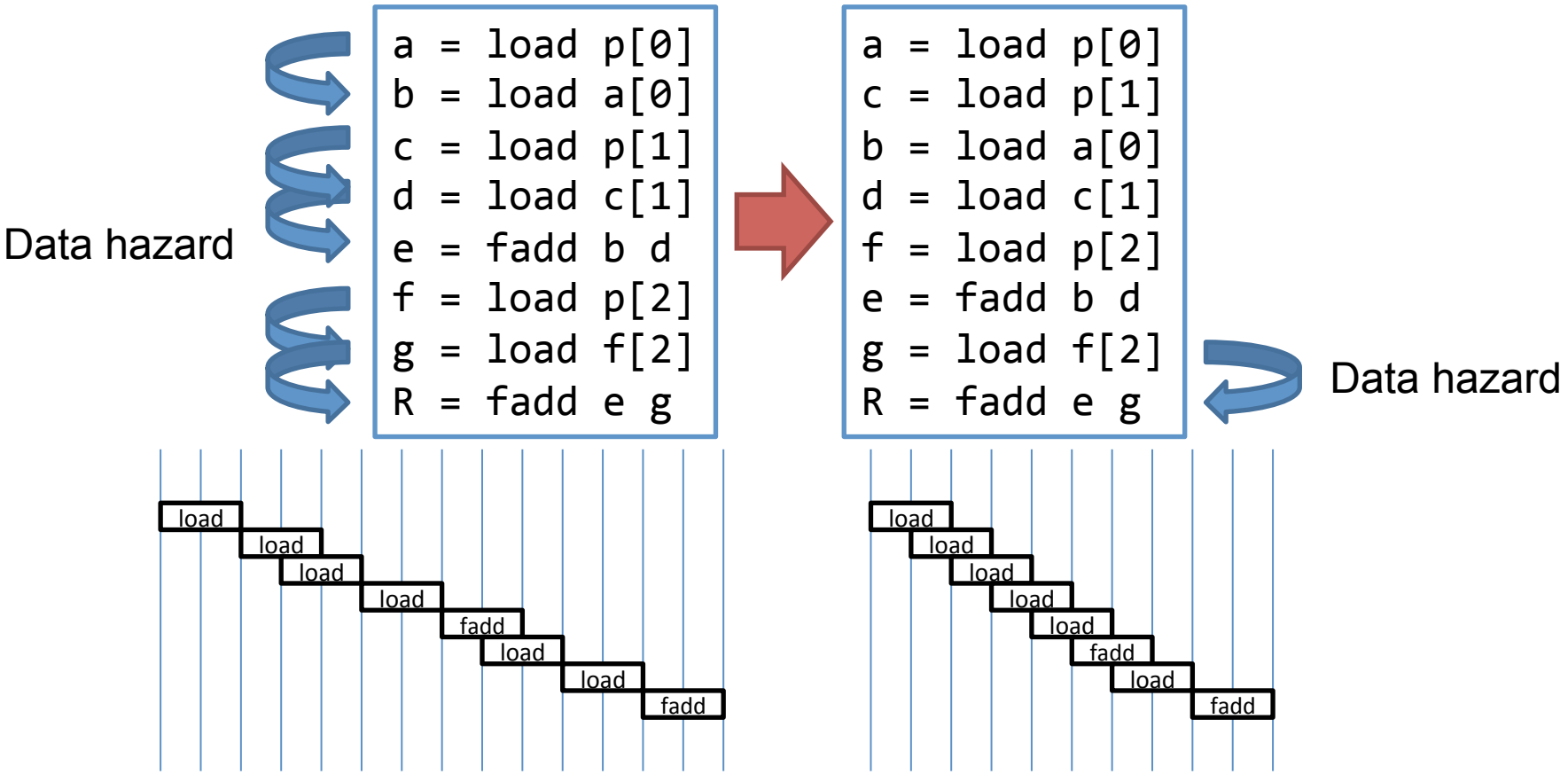
- 命令スケジューリング
- グラフ彩色によるレジスタ割り当て

命令スケジューリングとは

- 命令の順序を並び替える事
- 二つの効果がある
  1. 命令レベル並列性の向上
  2. データ局所性向上
     (→ レジスタ割り当ての効率向上)

命令レベル並列性の向上

例: p[0][0] + p[1][1] + p[2][2]



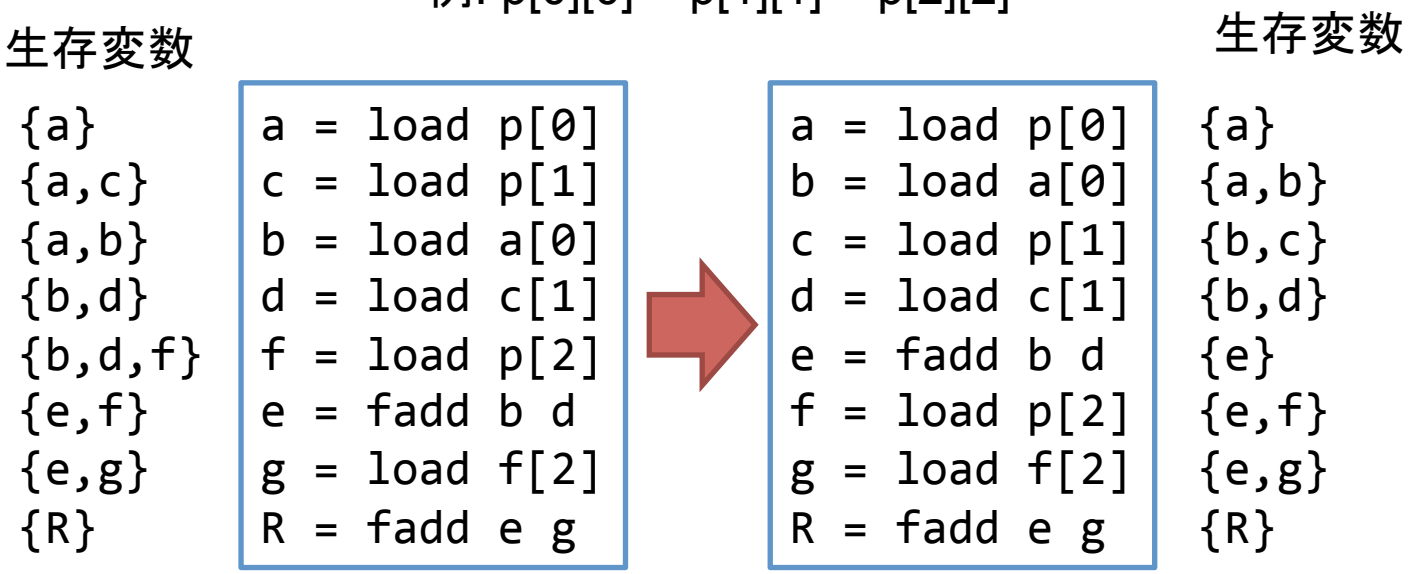
14 サイクル

10 サイクル

(load,faddのレイテンシが2の場合)

データ局所性の向上

例: p[0][0] + p[1][1] + p[2][2]



レジスタが 3 つ必要

レジスタは 2 つで良い

並列性と局所性のトレードオフ

- 命令レベル並列性を上げる為には...
  - 無関係な (因果関係にない) 命令を近くに配置する
- データの局所性を上げる為には...
  - 関係する (因果関係がある) 命令を近くに配置する

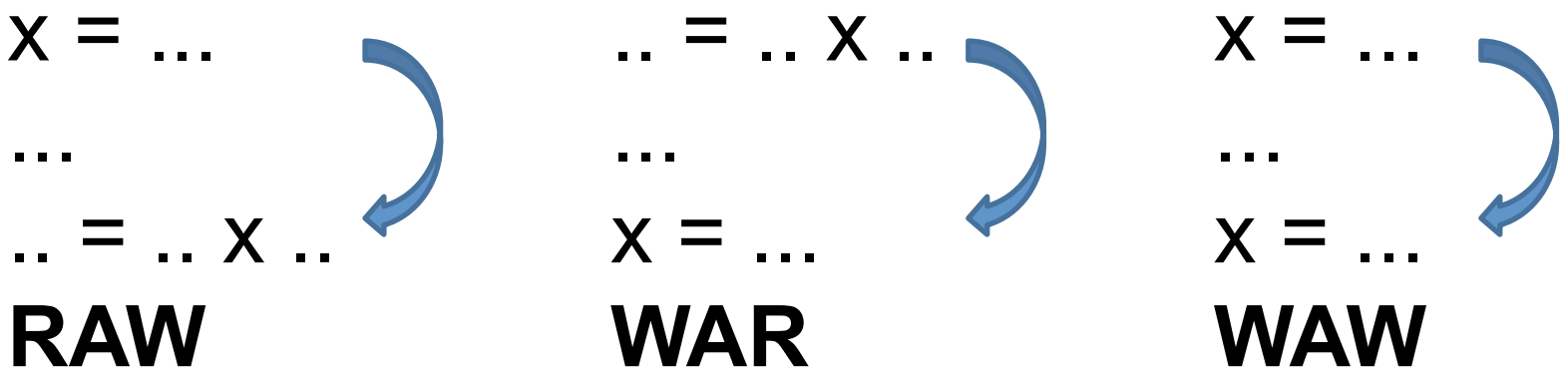
- どの様な戦略を取るべきかはアーキテクチャに依る

スケジューリング(リストスケジューリング)の手順

1. 命令間の依存を解析しグラフ構築
2. グラフから一命令づつ取り出しながらスケジュール

依存解析

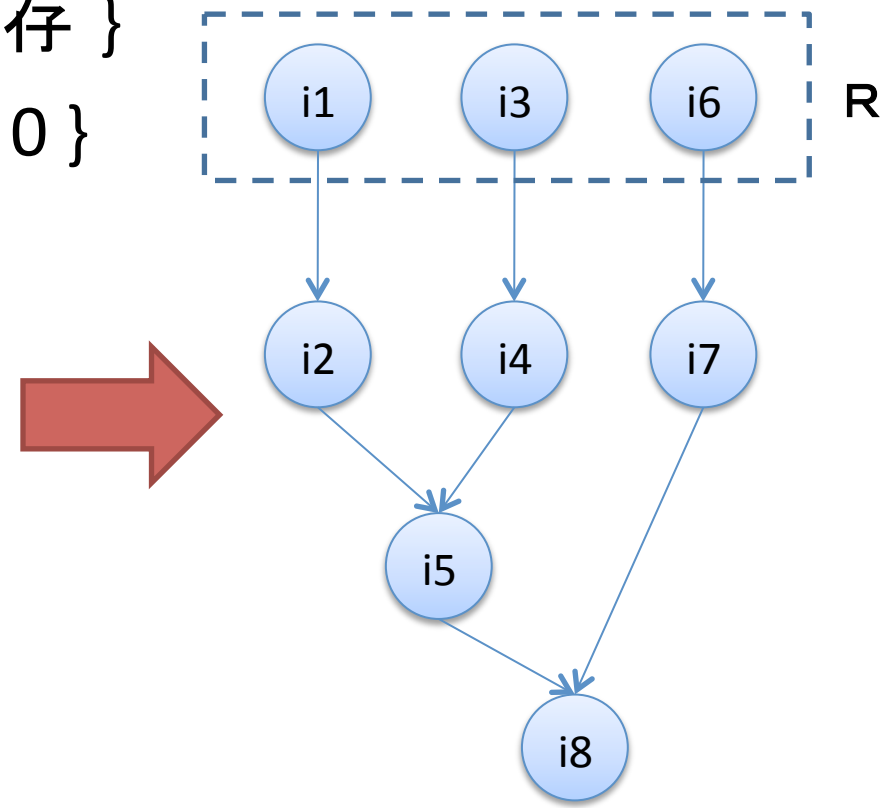
- 命令 a, bの順番を変えると意味が変わるとき「b は a に依存する」という。
  1. RAW (Read after Write) 依存関係
  2. WAR (Write after Read) 依存関係
  3. WAW (Write after Write) 依存関係



依存グラフと ready set

- 依存グラフ G = (V, E) と ready set R を構築
  - V = {命令}
  - E = { (a, b) | bがaに依存 }
  - R = { a | indegree(a) = 0 }

```
i1: a = load p[0]
i2: b = load a[0]
i3: c = load p[1]
i4: d = load c[1]
i5: e = fadd b d
i6: f = load p[2]
i7: g = load f[2]
i8: R = fadd e g
```



資源制約

- 資源制約: 演算器等が使用可能かどうか?
  - テーブルを作成して管理

cycle n

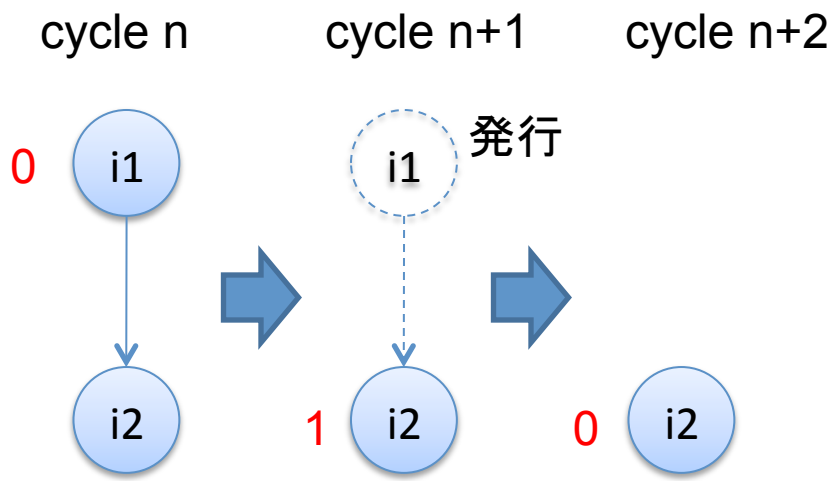
ALU	FADD	MEM	
	c = fadd a b	x = load p[0]	ステージ 1
			ステージ 2

cycle n+1

ALU	FADD	MEM	
			ステージ 1
	c = fadd a b	x = load p[0]	ステージ 2

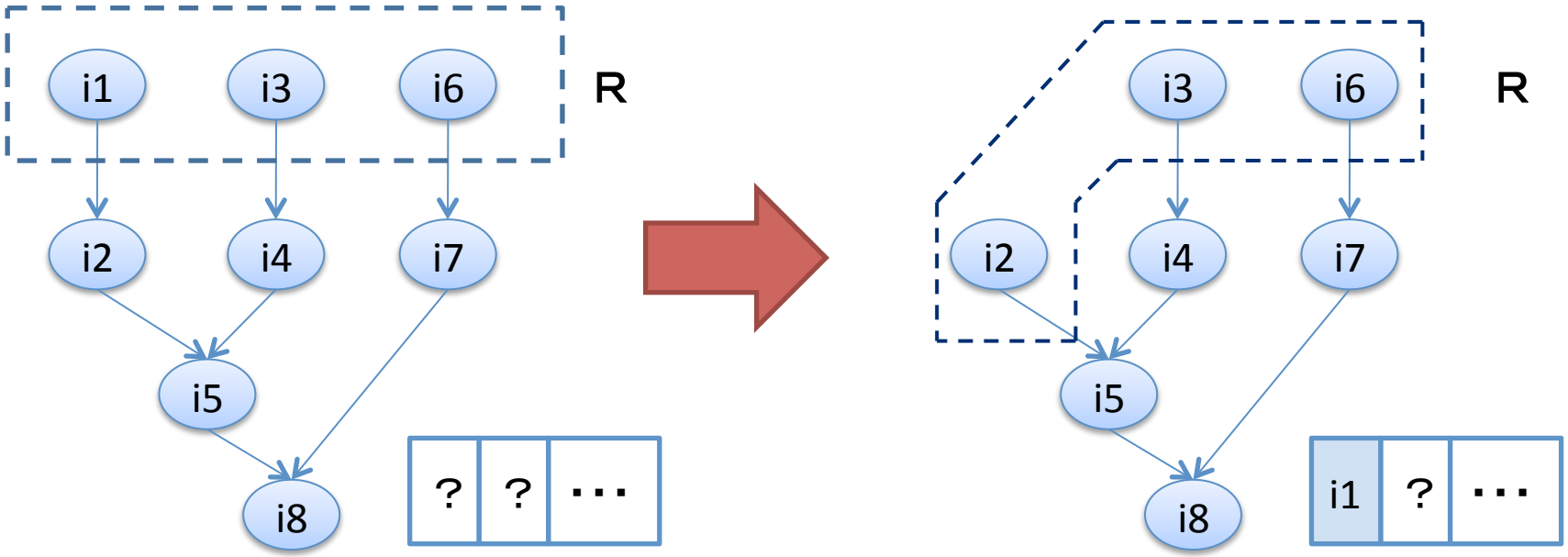
レイテンシ制約

- レイテンシ制約: 演算結果が使用可能かどうか?
  - Ready set の各命令に
    あと何サイクル待つ
    必要があるかを表す
    カウンタを付与



リストスケジューリング

- 全命令を並べ終わるまで以下を繰り返す。
  1. Ready set に実行可能な命令あれば
     優先度の高い命令を一つ取り出し並べる
  2. グラフ、ready set を更新
     - 資源制約・レイテンシ制約を 1 サイクル分更新



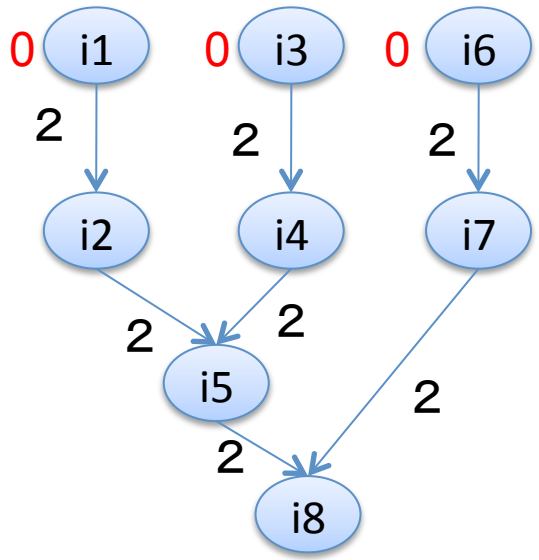
スケジューリングの戦略

- 何を優先的に取り出すか?
  - 実行時間を優先 vs 資源節約を優先
  - 「優先度が高いが制約を満たさない命令」と「優先度は低いが制約を満たす命令」のどちらを先に並べるべきか?
- どれくらい真面目に計算するか?
  - 最適なスケジューリングを行う事は一般に NP 困難

実行時間優先のスケジューリング (例)

- クリティカルパスを優先的にスケジュール
  - 先行命令のレイテンシを辺の重みとする

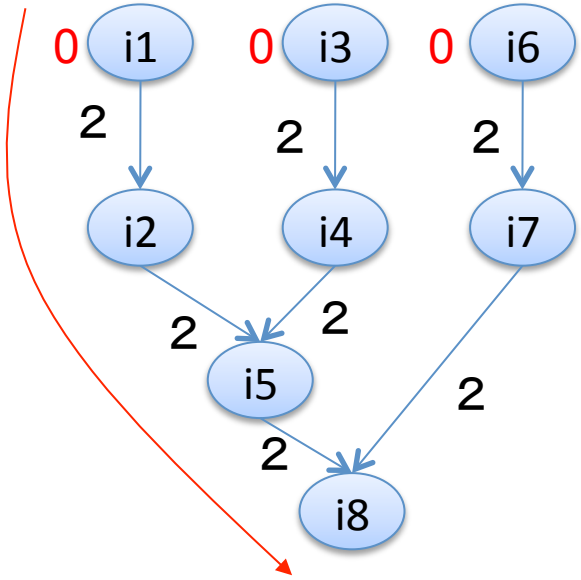
cycle = 0



ALU	FADD	MEM

実行時間優先のスケジューリング (例)

- クリティカルパスを優先的にスケジュール



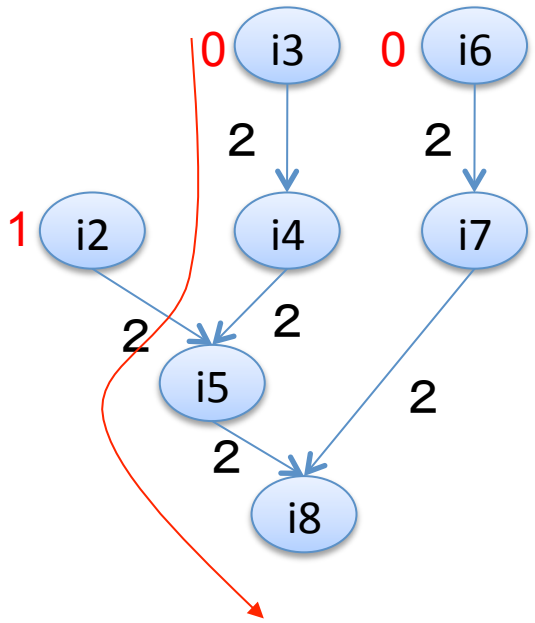
cycle = 0

ALU	FADD	MEM
		i1

i1

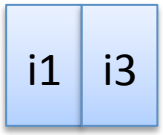
実行時間優先のスケジューリング (例)

- クリティカルパスを優先的にスケジュール



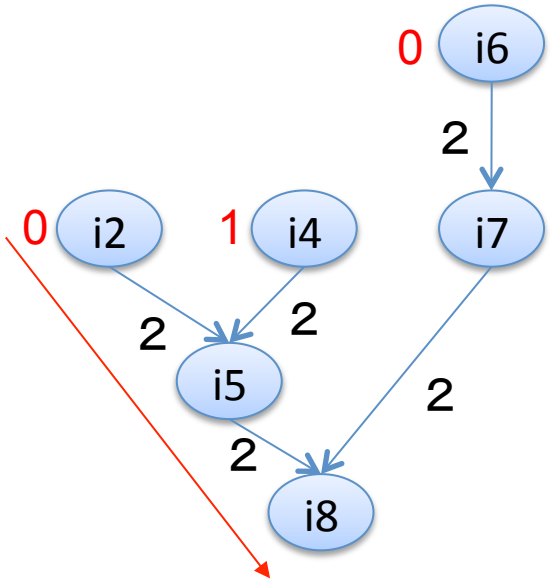
cycle = 1

ALU	FADD	MEM
		i3
		i1



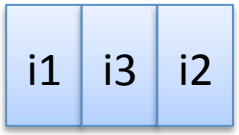
実行時間優先のスケジューリング (例)

- クリティカルパスを優先的にスケジュール



cycle = 3

ALU	FADD	MEM
		i2
		i3

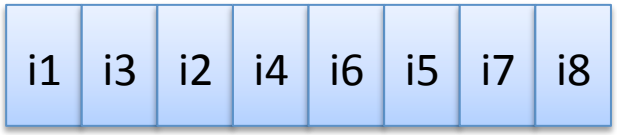


実行時間優先のスケジューリング (例)

- クリティカルパスを優先的にスケジュール

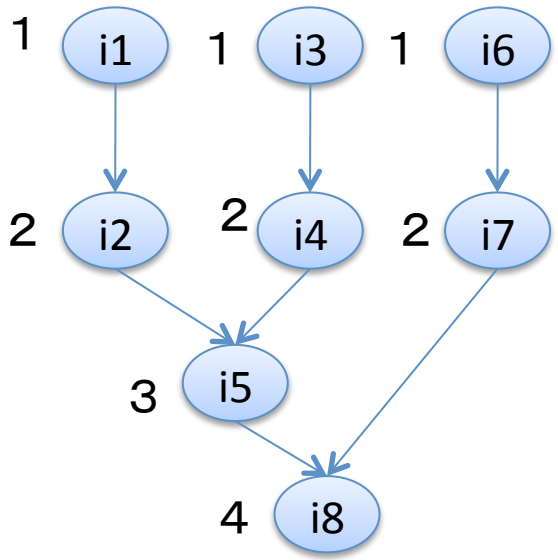
```
i1: a = load p[0]
i3: c = load p[1]
i2: b = load a[0]
i4: d = load c[1]
i6: f = load p[2]
i5: e = fadd b d
i7: g = load f[2]
i8: R = fadd e g
```

10 サイクル
レジスタ 3 個



資源節約優先のスケジューリング (例)

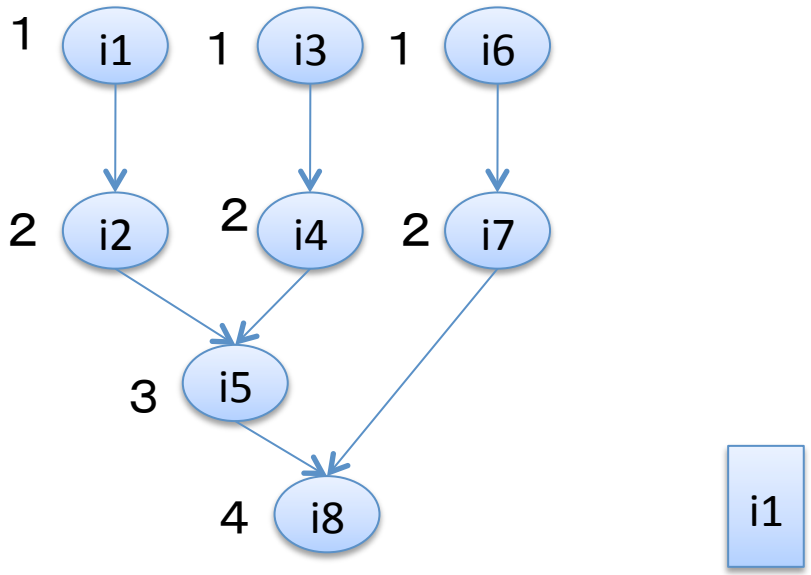
- 既に並べた命令に依存する命令を
  優先的にスケジュール



後続命令に先行命令より高い優先度を付与

資源節約優先のスケジューリング (例)

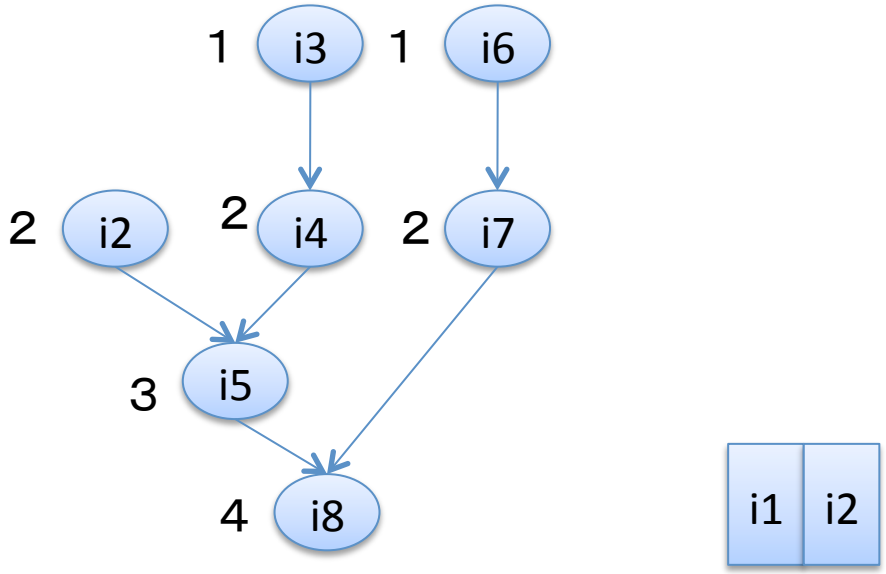
- 既に並べた命令に依存する命令を
  優先的にスケジュール



後続命令に先行命令より高い優先度を付与

資源節約優先のスケジューリング (例)

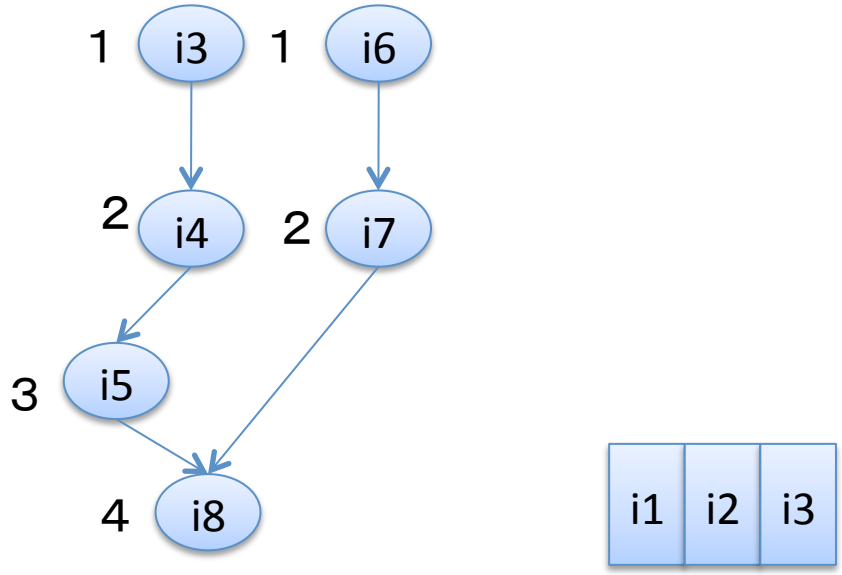
- 既に並べた命令に依存する命令を
  優先的にスケジュール



後続命令に先行命令より高い優先度を付与

資源節約優先のスケジューリング (例)

- 既に並べた命令に依存する命令を
  優先的にスケジュール



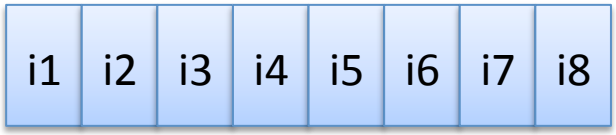
後続命令に先行命令より高い優先度を付与

資源節約優先のスケジューリング (例)

- 既に並べた命令に依存する命令を優先的にスケジュール

```
i1: a = load p[0]
i2: b = load a[0]
i3: c = load p[1]
i4: d = load c[1]
i5: e = fadd b d
i6: f = load p[2]
i7: g = load f[2]
i8: R = fadd e g
```

14 サイクル
レジスタ 2 個



後続命令に先行命令より高い優先度を付与

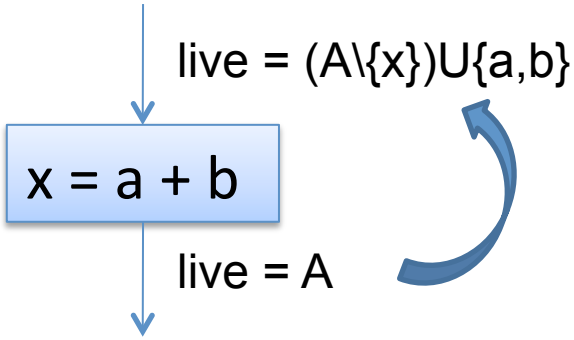
グラフカラーリングによる
レジスタ割り当て

- 現実のコンパイラで幅広く用いられている方法
- 手順
  1. 生存解析
  2. 干渉グラフ構築
  3. グラフ塗り分け

生存解析

- 各命令実行直後に生きている変数を求める
  - 「生きている ⇔ その後使われる」なので後方解析
  - live[i]: 命令iの実行直後に生きている変数
  - def[i]: 命令iが定義する変数
  - use[i]: 命令iが使用する変数

$$live[i] = \bigcup_{j \in succ(i)} (live[j] \setminus def[j]) \cup use[j]$$



生存解析の例

例 : m[i]*m[j]/(x[i]-x[j])^2

```
a = load m[i]
b = load m[j]
d = load x[i]
c = fmul a b
e = load x[j]
f = fsub d e
g = fmul f f
R = fdiv c g
ret
```

R: 戻り値用レジスタ

live

生存解析の例

例 : m[i]*m[j]/(x[i]-x[j])^2

```
a = load m[i]
b = load m[j]
d = load x[i]
c = fmul a b
e = load x[j]
f = fsub d e
g = fmul f f
R = fdiv c g
ret
```

{R}

R: 戻り値用レジスタ

live

生存解析の例

例 : m[i]*m[j]/(x[i]-x[j])^2

```
a = load m[i]
b = load m[j]
d = load x[i]
c = fmul a b
e = load x[j]
f = fsub d e
g = fmul f f
R = fdiv c g
ret
```

```
{R}        # ({R}＼Φ)UΦ
{R}
```

R: 戻り値用レジスタ

live

生存解析の例

例 : m[i]*m[j]/(x[i]-x[j])^2

```
a = load m[i]
b = load m[j]
d = load x[i]
c = fmul a b
e = load x[j]
f = fsub d e
g = fmul f f
R = fdiv c g
ret
```

```
{c,g}   # ({R}＼{R})U{c,g}
{R}
{R}
```

R: 戻り値用レジスタ

live

生存解析の例

例：m[i]*m[j]/(x[i]-x[j])^2

```
a = load m[i]
b = load m[j]
d = load x[i]
c = fmul a b
e = load x[j]
f = fsub d e
g = fmul f f
R = fdiv c g
ret
```

```
{c,f}   # ({c,g}＼{g})U{f}
{c,g}
{R}
{R}
```

R: 戻り値用レジスタ

live

生存解析の例

例：m[i]*m[j]/(x[i]-x[j])^2

```
a = load m[i]
b = load m[j]
d = load x[i]
c = fmul a b
e = load x[j]
f = fsub d e
g = fmul f f
R = fdiv c g
ret
```

```
{c,d,e} # ({c,f}\{f})U{d,e}
{c,f}
{c,g}
{R}
{R}
```

R: 戻り値用レジスタ

live

生存解析の例

例：m[i]*m[j]/(x[i]-x[j])^2

	live
	{i,j,m,x}
a = load m[i]	{a,i,j,m,x}
b = load m[j]	{a,b,i,j,x}
d = load x[i]	{a,b,d,j,x}
c = fmul a b	{c,d,j,x}
e = load x[j]	{c,d,e}
f = fsub d e	{c,f}
g = fmul f f	{c,g}
R = fdiv c g	{R}
ret	{R}

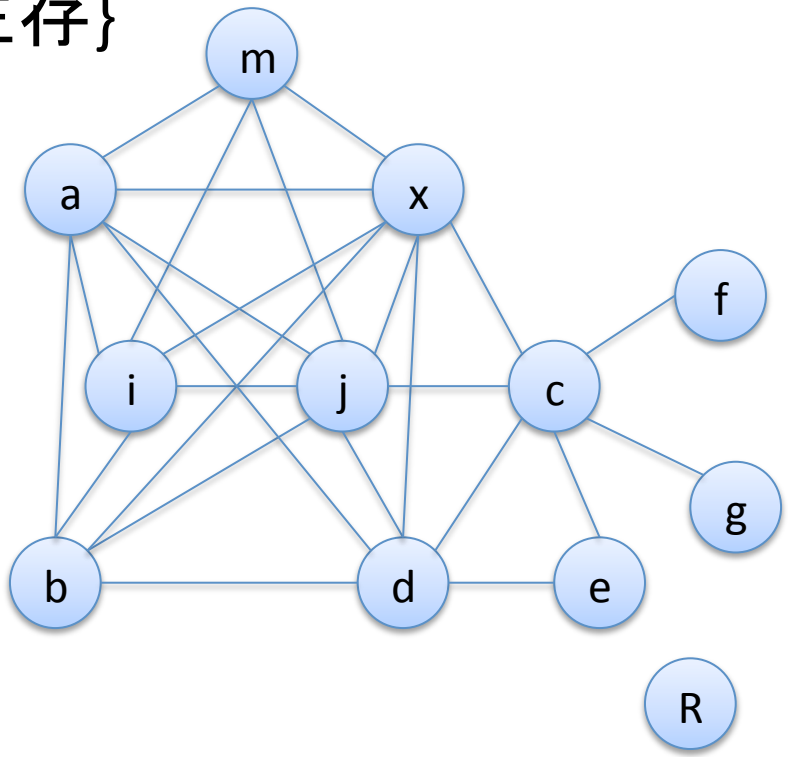
R: 戻り値用レジスタ

干渉グラフ

- G = (V, E)
  - V = {変数 or レジスタ}
  - E = {(a, b) | a,bが同時に生存}

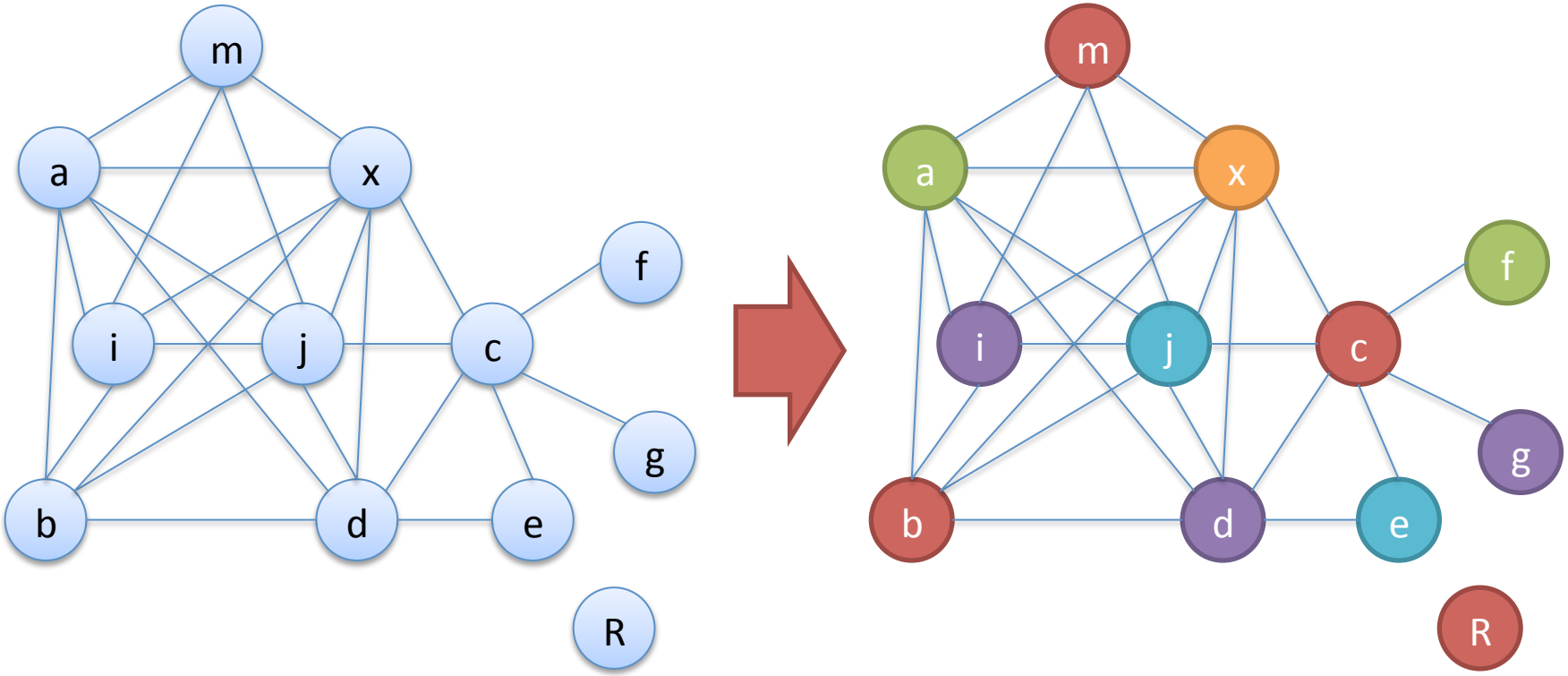
```
a = load m[i]
b = load m[j]
d = load x[i]
c = fmul a b
e = load x[j]
f = fsub d e
g = fmul f f
R = fdiv c g
ret
```

```
{i,j,m,x}
{a,i,j,m,x}
{a,b,i,j,x}
{a,b,d,j,x}
{c,d,j,x}
{c,d,e}
{c,f}
{c,g}
{R}
{R}
```



レジスタ割り当て

- レジスタ割り当て
  ⇒ 干渉グラフのカラーリング

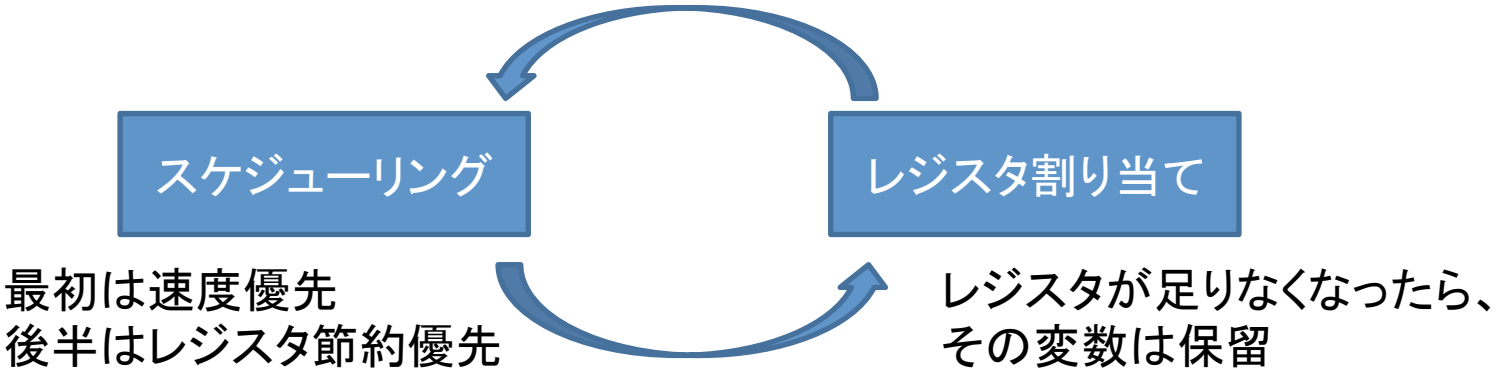


カラーリングの方法

- 色数が最小となる最適カラーリング
  を求める事は NP 困難
- 各変数が spill した場合のコストを計算し
  それが大きい順に割り当てる
  - Spill コストの大きい変数
    - ループ内変数
    - 複数回参照される変数
    - 同一命令中で使われる他の変数が spill している
      - 普通、一命令中で複数のメモリアクセスはできない

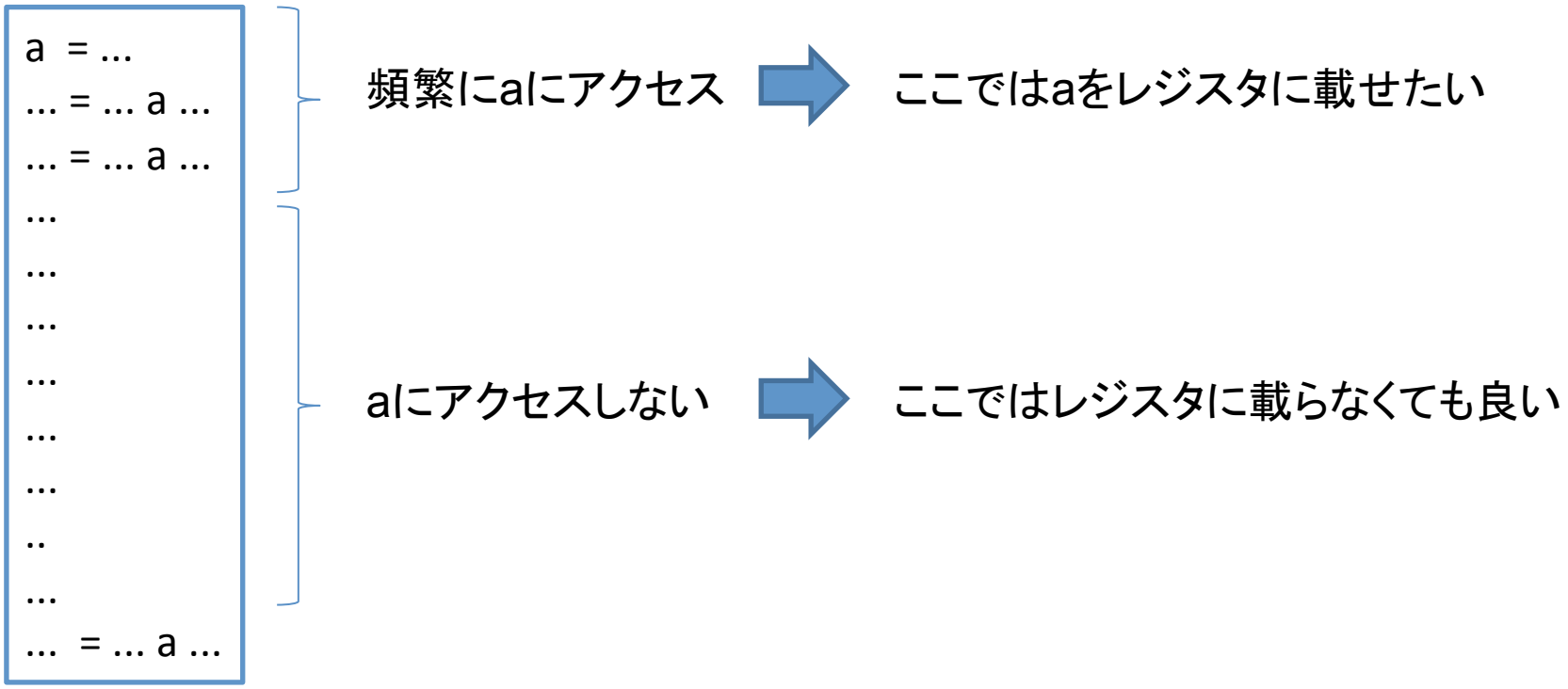
スケジューリングとレジスタ割り当て: 全体の流れ

- レジスタ数に余裕があるなら
  - スケジューリング → レジスタ割り当て
- ないなら
  - レジスタ割り当て→スケジューリング
- 微妙なら交互に



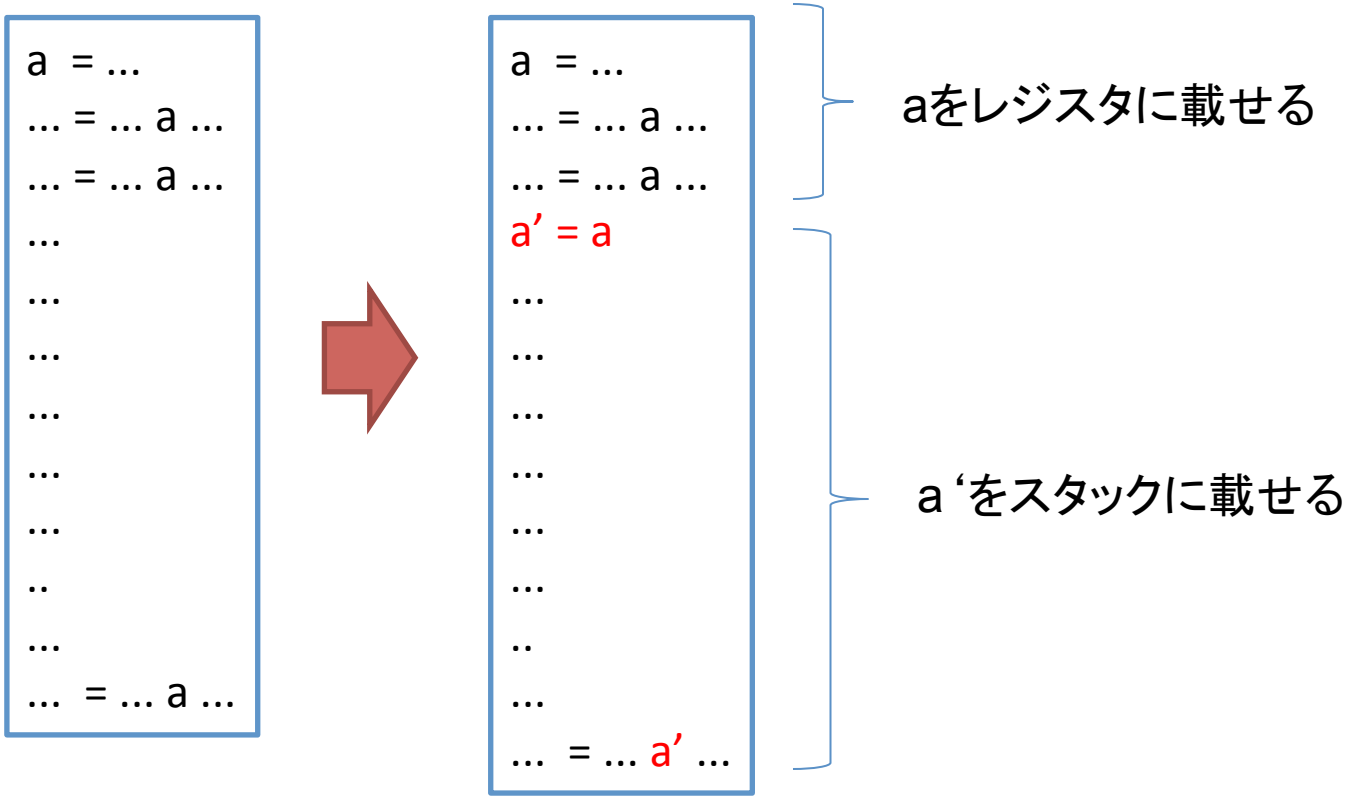
生存区間分割

- 長く生きている変数の名前を途中で替える



生存区間分割

- 長く生きている変数の名前を途中で替える



共通課題

- 二つの共通課題のうち一つ以上を解いてください

課題1

- リストスケジューリングを実装せよ
  - どのような優先順位を設定したか説明すること
  - 最低2種類の戦略を実装し、比較・評価をすること

課題2

- min-rt.ml から適度に大きい関数を選び
  手作業でレジスタ割り当てと
  スケジューリングをした結果を示せ
  - 何らかのアルゴリズムに従って行うこと

- 各班の自作コンパイラの出力を用いてもよいし
  MinCaml の出力を用いてもよい。

コンパイラ係向け課題

- グラフカラーリング
  (もしくはそれに準ずるアルゴリズム)
  によるレジスタ割り当てを実装せよ
  - どのような塗り分け方法を採ったか
    選択の理由を含めて説明せよ

課題の提出先と締め切り

- 提出先: compiler-report-2011@yl.is.s.u-tokyo.ac.jp
- 締め切り: 2 週間後 (12/22) の午後 1 時 (JST)
  - コンパイラ係向け課題締切 : 2012/2/27
- Subject: **`Report␣9␣`**<学籍番号: 5 桁>

半角スペース 1 個ずつ

  - 例: **`Report 9 11099`**
- 本文にも氏名と学籍番号を明記のこと

◆ 質問は compiler-query-2011@yl.is.s.u-tokyo.ac.jp まで