入出力，OS，計算機の高速化

これまで何を学んだか

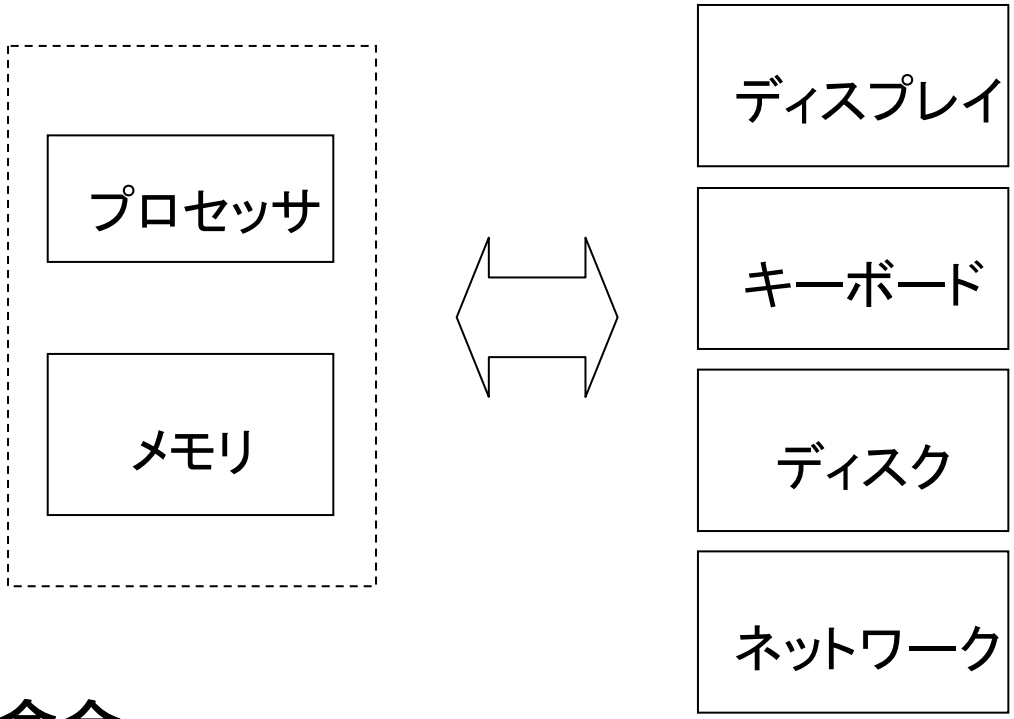
- 0 と 1 の世界
  - 2進数，算術演算，論理演算，浮動小数点数
- 計算機はどのように動くのか
  - プロセッサとメモリ
  - 演算命令，ロード・ストア命令，分岐命令
- 計算機はどのように構成されているのか
  - 組合せ回路 ≒ 論理関数
    - 論理式の標準形，論理式の簡単化
  - 順序回路 ≒ 有限状態機械
  - メインメモリ，キャッシュメモリ

目次

- 「プロセッサとメモリ」以外
- オペレーティングシステム
  - どのように複数のプログラムが動くのか

- 計算機の高速化
  - 計算機アーキテクチャはどのように進化してきたか

入出力 (I/O)



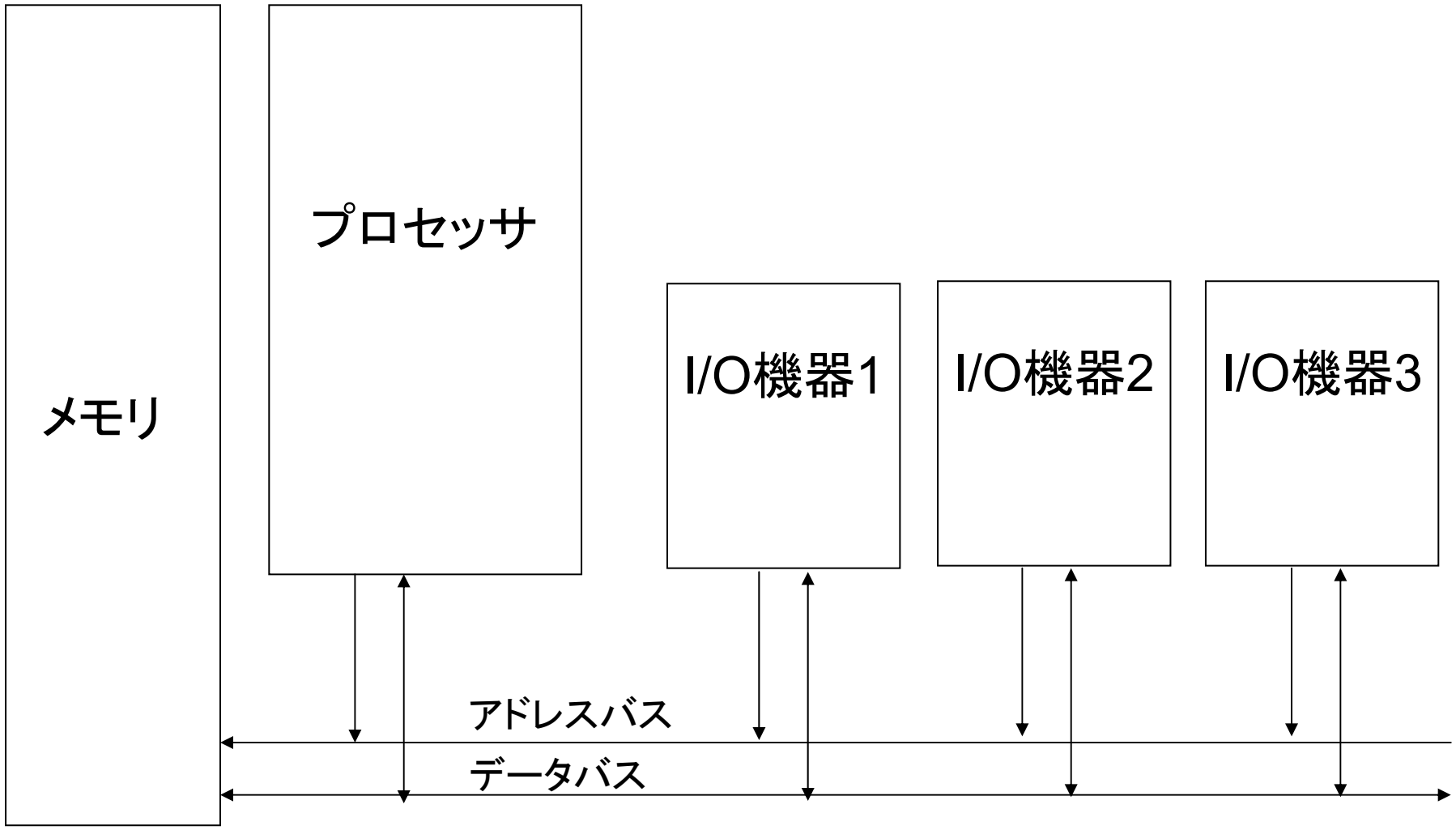
入出力専用命令

入出力装置に入力・出力するための専用の命令が用意されている（機器ごとに入出力アドレス）x86 はこの方式

メモリマップ入出力

ある特定のメモリアドレスを読み書きすると，入力・出力が行われる（機器ごとに異なるメモリアドレス）MIPS はこの方式

メモリマップトI/O



例題

多くの場合，入出力機器にデータを要求してから実際にそれが得られるまでにかかる時間を予測するのは難しい．データが得られるまで入力命令を繰り返し実行することをポーリングと呼ぶ．

最大 4 メガバイト/秒のレートで 4 バイトずつのデータをプロセッサに送ってくる入力機器があるとする．ポーリングによってこのデータを取りこぼしなく受け取りたい．ポーリング 1 回に 100 クロックサイクルが必要であるとし，プロセッサのクロック周波数は 1 GHz とする．実行時間のうちポーリングに費やされる割合を求めよ．

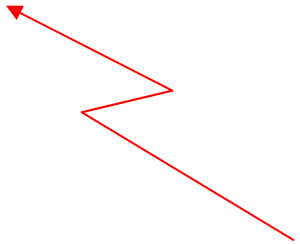
（解答例）取りこぼしを防ぐには 1 秒に 1000^2 回以上のポーリングが必要である．1 秒間のクロックサイクル数 1000^3 のうち $100 \leq 1000^2$ サイクルがポーリングに費やされるので，その割合は 10 % である．

割込み

プログラム

命令
命令
入力命令
命令
出力命令
…

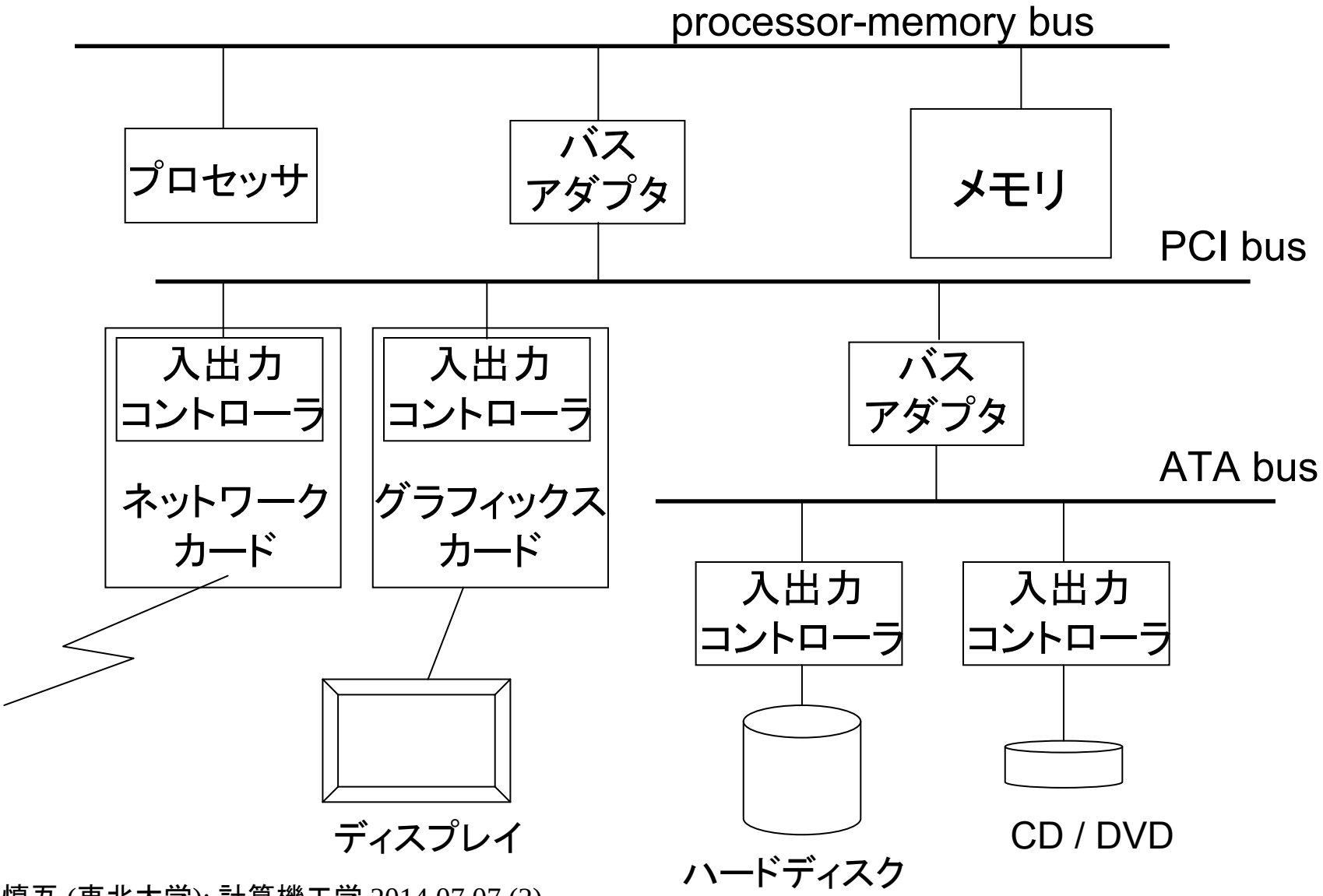
割込ハンドラ1:
命令
命令
復帰



割込信号1

機器側からプロセッサへ，何らかの事象の発生を知らせる
（e.g. データの準備ができた．ネットワークから情報が届いた）

バス


processor-memory bus
プロセッサ
バス
アダプタ
メモリ
PCI bus
入出力
コントローラ
ネットワーク
カード
入出力
コントローラ
グラフィックス
カード
バス
アダプタ
ATA bus
入出力
コントローラ
入出力
コントローラ
ディスプレイ
ハードディスク
CD / DVD

オペレーティングシステム (OS)

「基本ソフトウェア」などと呼ばれる

- Windows
- MacOS
- Linux
- Solaris
- …

} UNIX 系

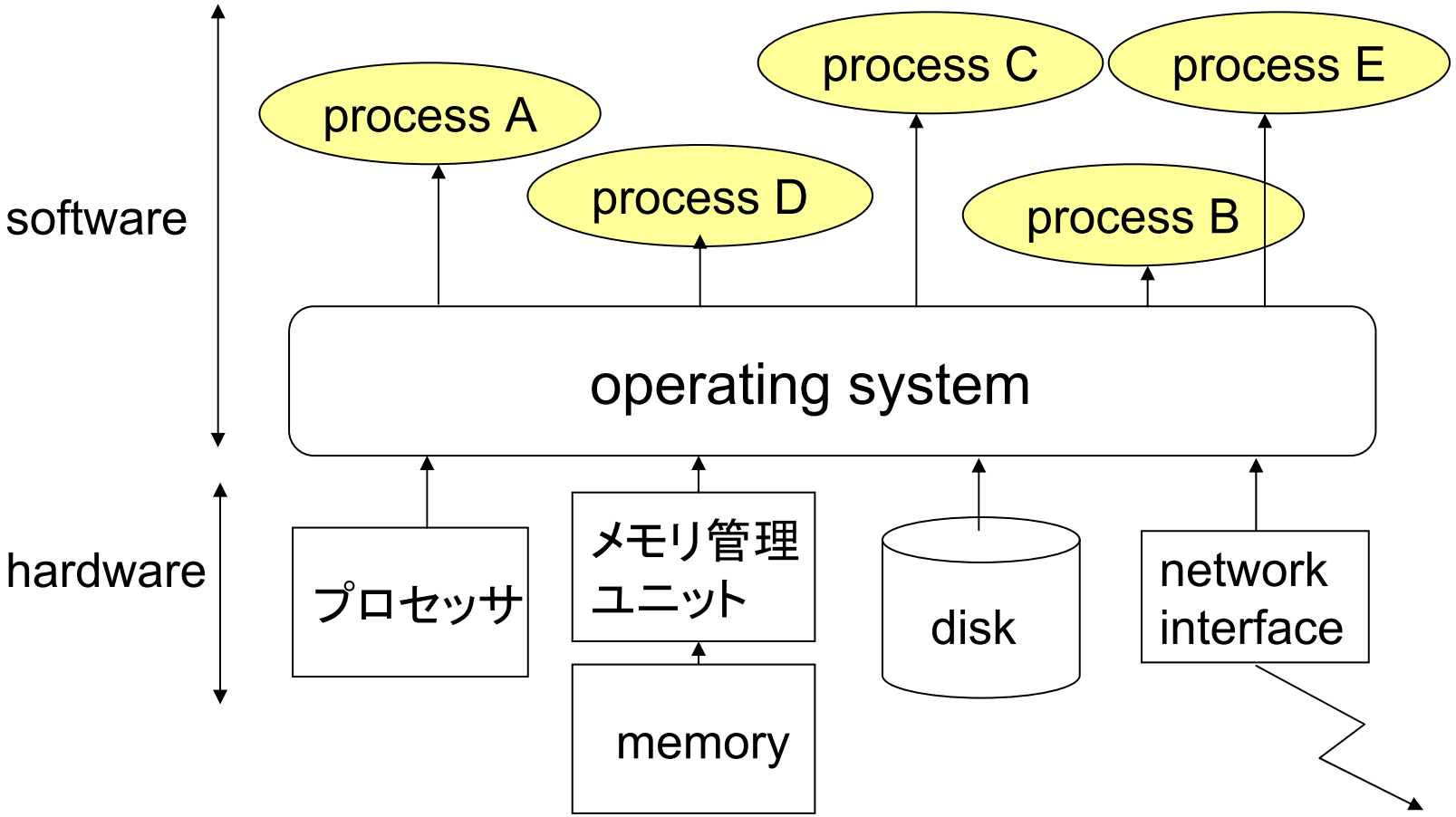
- ハードウェアの詳細を隠蔽して，抽象化されたマシンをプログラムに提供する

例: A社のハードディスク，B社のハードディスク，C社のUSBメモリ
→「ファイル」という概念で統一的に操作できる

- 複数のプログラム，複数のユーザの間で，必要な資源（ハードウェア）を適切に管理する

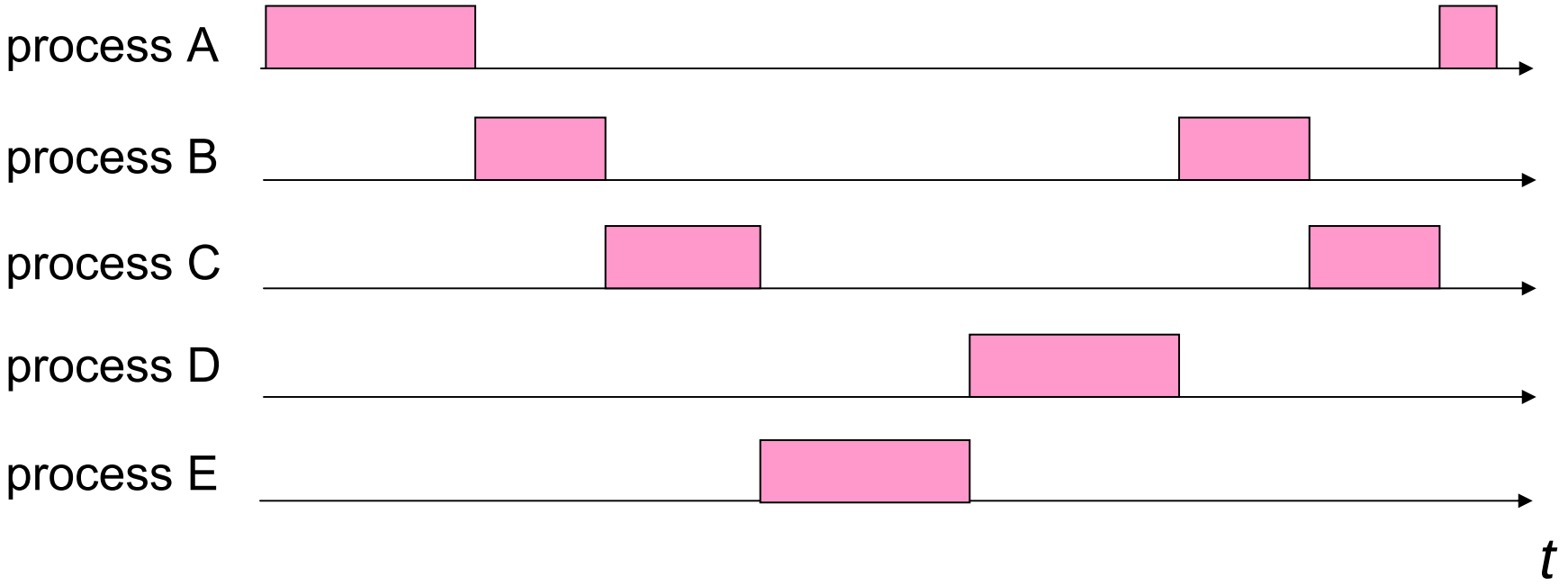
例: 同時にディスクを読み書きしても大丈夫
Word が不正なアドレスのメモリを読み書きしても，Excel には影響がない

OS の概念

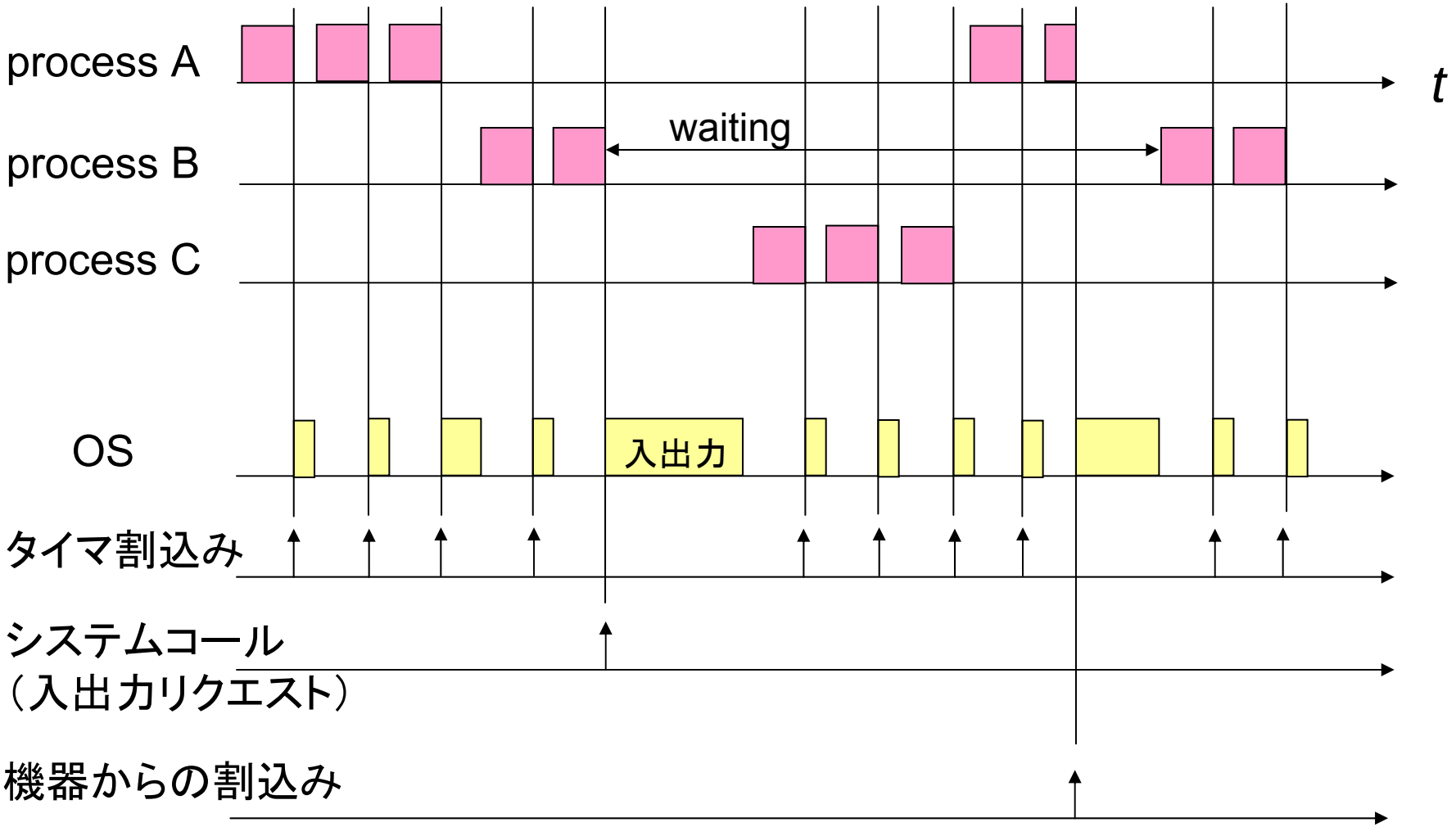


複数のプロセスにハードウェア資源を（多くの場合**時分割**で）
割り当てるソフトウェア

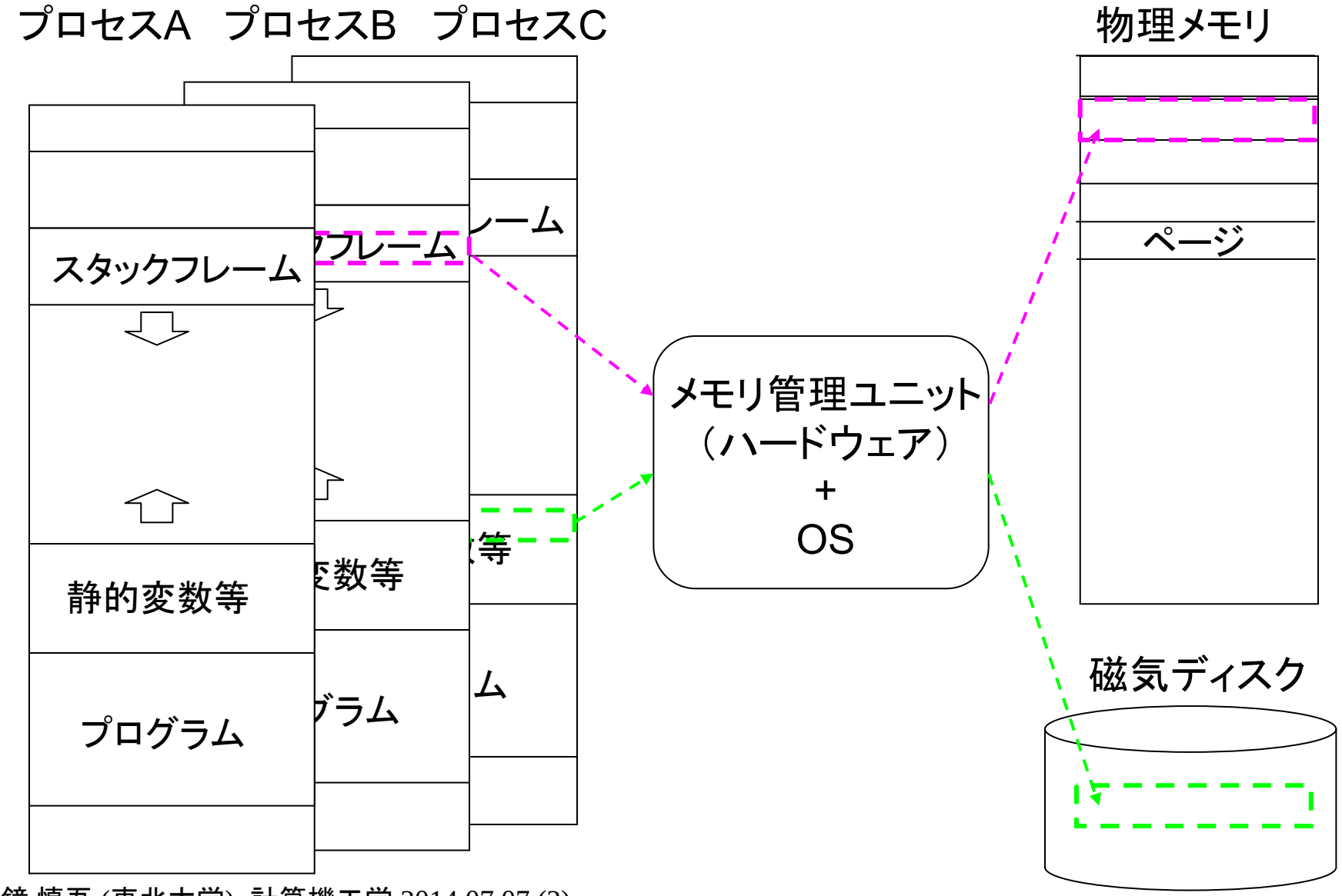
時分割処理



プロセッサ時間の割り当て



仮想記憶


プロセスA　プロセスB　プロセスC
スタックフレーム
静的変数等
プログラム
メモリ管理ユニット
（ハードウェア）
+
OS
物理メモリ
ページ
磁気ディスク

目次

- 「プロセッサとメモリ」以外
- オペレーティングシステム
  - どのように複数のプログラムが動くのか

- 計算機の高速化
  - 計算機アーキテクチャはどのように進化してきたか

計算機の性能

多くの場合，プログラムの「平均」実行時間が指標となる

$$\text{実行時間} = \text{クロックサイクル時間} \times \text{実行命令数} \times \text{平均CPI}$$
$$= \text{クロックサイクル時間} \times \Sigma_i (\text{実行命令数}_i \times \text{CPI}_i)$$

- CPI (clock cycle per instruction): 命令あたりのクロックサイクル数
- 実行命令数$_i$： 命令 i の実行回数
- CPI$_i$: 命令 i の CPI

一般に，クロックサイクル時間・実行命令数・CPIの相互間にはトレードオフの関係がある

大原則: 実行時間全体を改善するためには，頻繁に実行される命令を優先して改善すべきである

計算機のコスト

製造コストは，同じ回路を実現するのに必要な面積が大きいと高くなる

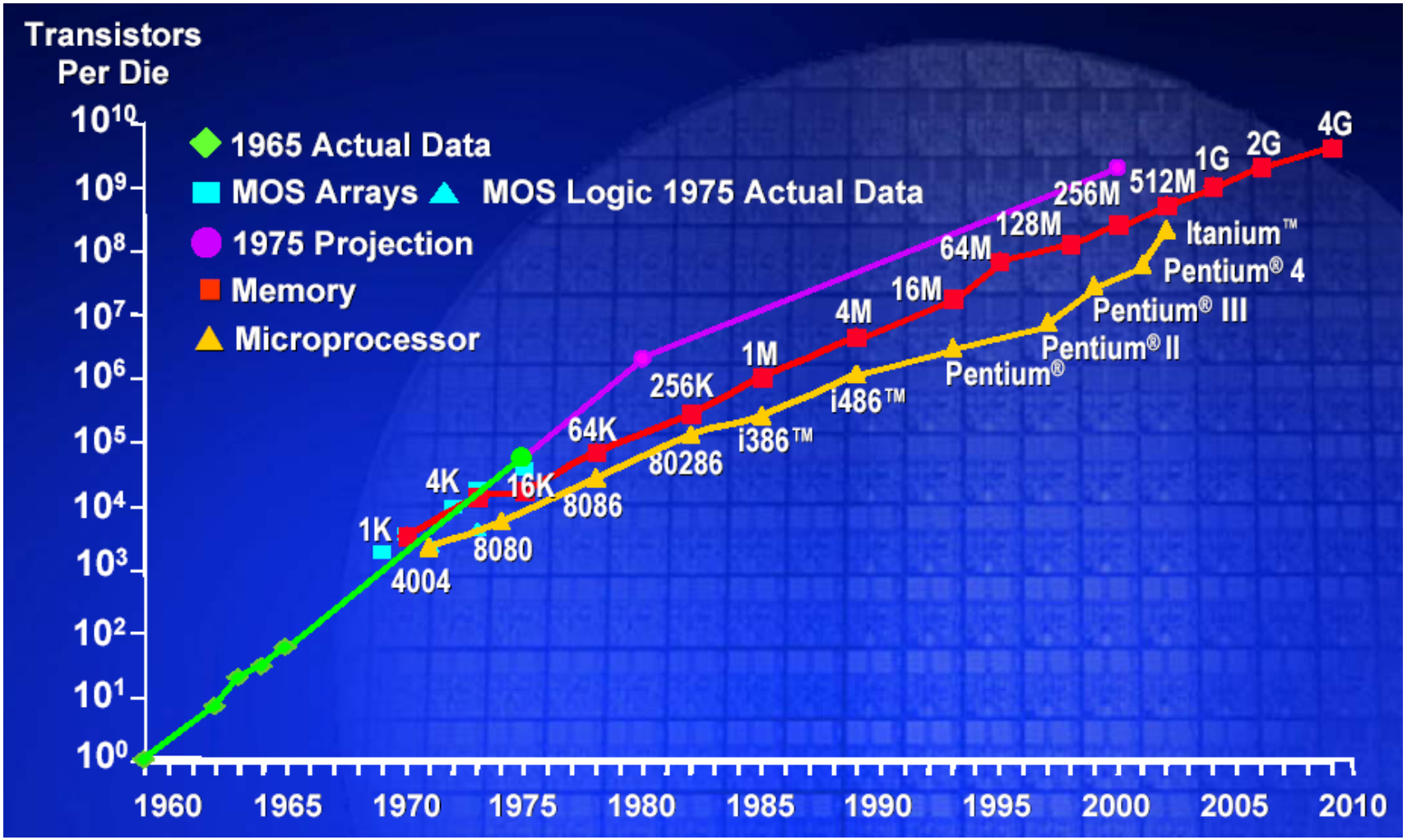
- 少なくとも比例するのは容易に想像できる
- 製造欠陥を考慮すると，実際には比例以上

Moore の法則

集積回路に乗る素子の数は，1～2年で2倍になる
(初出: G. Moore: Electronics, Vol.38, No.8, 1965)
そしておおむねその通りになってきた

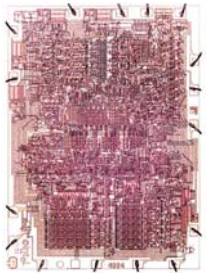
さらに固定費: 設計開発，設備投資

Moore's Law



Intel のデータ [Moore ISSCC2003]

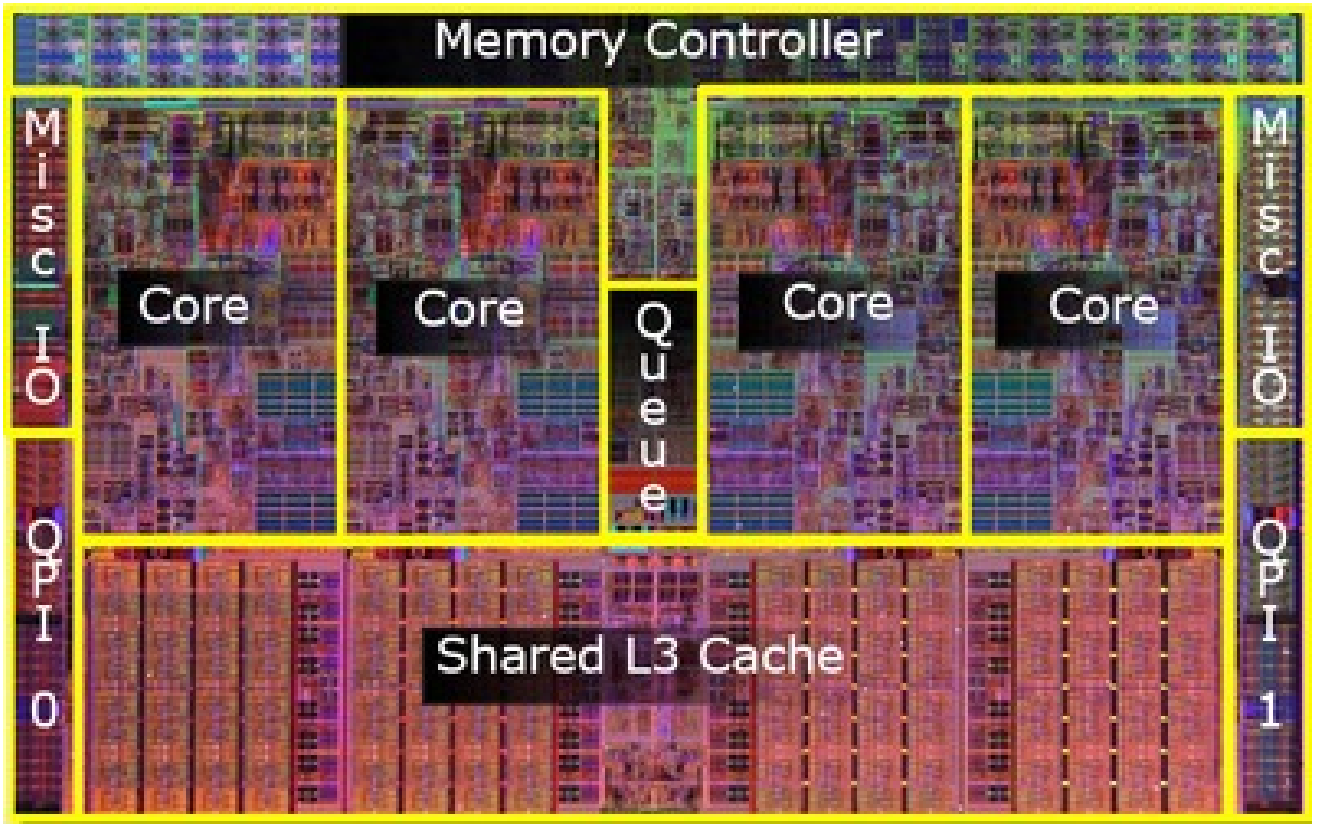
Intel 4004 (1971)



2300 トランジスタ
12 mm^2

cf.
1 D-FF = 12 Tr.
2300 / 12 ≒ 192
192 / 32 = 6

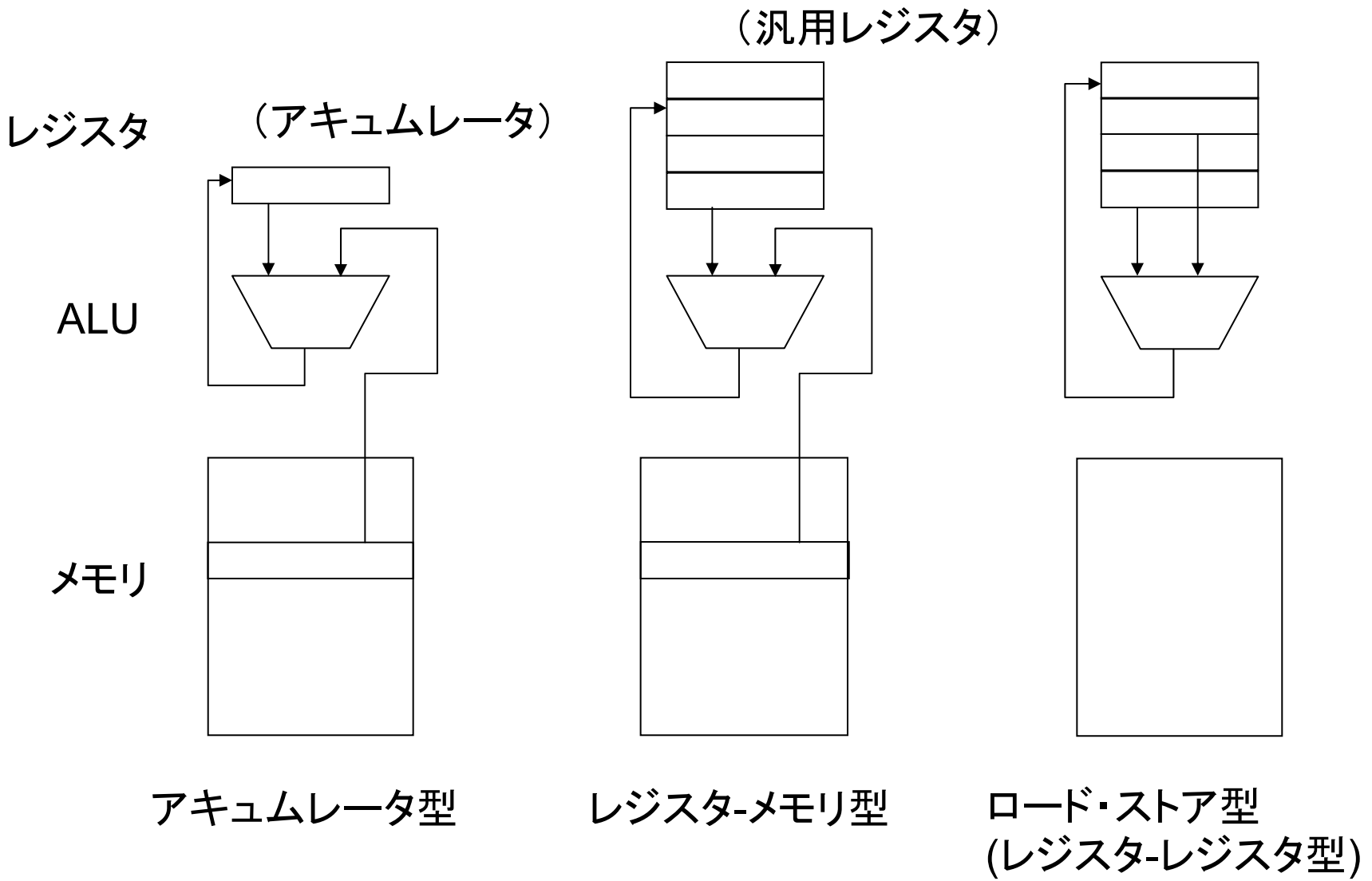
Intel Core i7 (2008)



731,000,000 トランジスタ
263 mm^2

http://news.com.com/1971+Intel+4004+processor/2009-1006_3-6038974-3.html
http://www.sharkyextreme.com/hardware/cpu/article.php/3782516/Intel-Core-i7-965-XE--Core-i7-920-Review.htm

命令セットアーキテクチャのいろいろ



MIPS以外の命令セットの例

6502	OP 10,Ri	# A ← A (op) mem[Ri + 10]
Z80	OP A, Rs	# A ← A (op) Rs
	OP A, (Ri+10)	# A ← A (op) mem[Ri + 10]
x86	OP Rd, Rs	# Rd ← Rd (op) Rs
	OP Rd, [Ri+10]	# Rd ← Rd (op) mem[Ri + 10]
MC680x0	OP Rs, Rd	# Rd ← Rd (op) Rs
	OP (10,Ri), Rd	# Rd ← Rd (op) mem[Ri + 10]
PowerPC / ARM	OP Rd, Rs1, Rs2	# Rd ← Rs1 (op) Rs2
SuperH	OP Rs, Rd	# Rd ← Rd (op) Rs

命令の大規模化・複雑化

メモリが高価で低速な時代: 命令読み出しが律速
　→ できるだけ命令数を少なくしたい

1個の命令で複雑な処理を実行できるようにすれば，同じ仕事を少ない命令数で実行できる

例1) MIPS で整数の配列にアクセスするには，配列インデックスを4倍し，配列先頭のアドレスに加算してからメモリ読み書き命令を実行する必要があった．命令セットによっては，これらを1命令で実行できる

例2) 多項式計算を1命令で実行できる命令セット

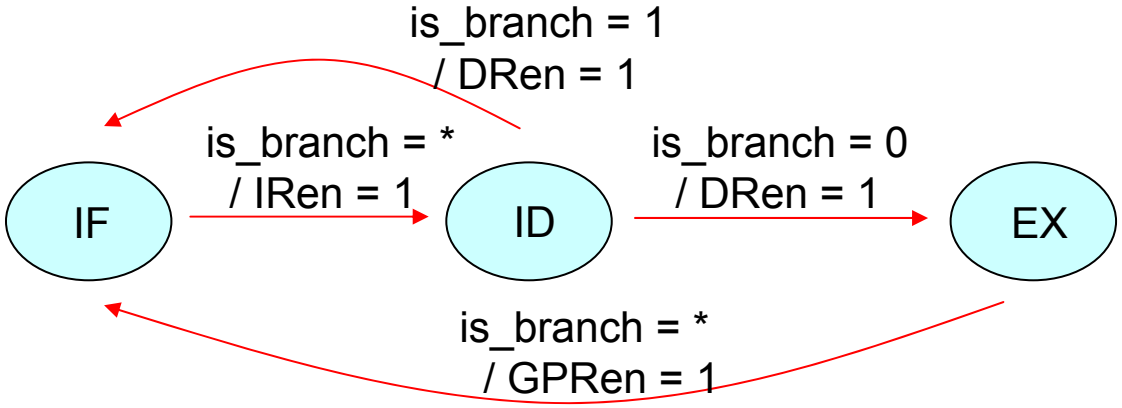
例3) 指定メモリ範囲からの値探索，指定メモリ範囲どうしのデータコピー

プロセッサの制御方式

- 授業で述べたような方式を布線論理制御 (hard wired logic control) と呼ぶ
- プロセッサの制御を，その状態遷移を「実行」する小さなプログラムによって行う方式をマイクロプログラム制御 (microprogrammed control) と呼ぶ
  - 状態遷移表の各行を「マイクロ命令」だと考える

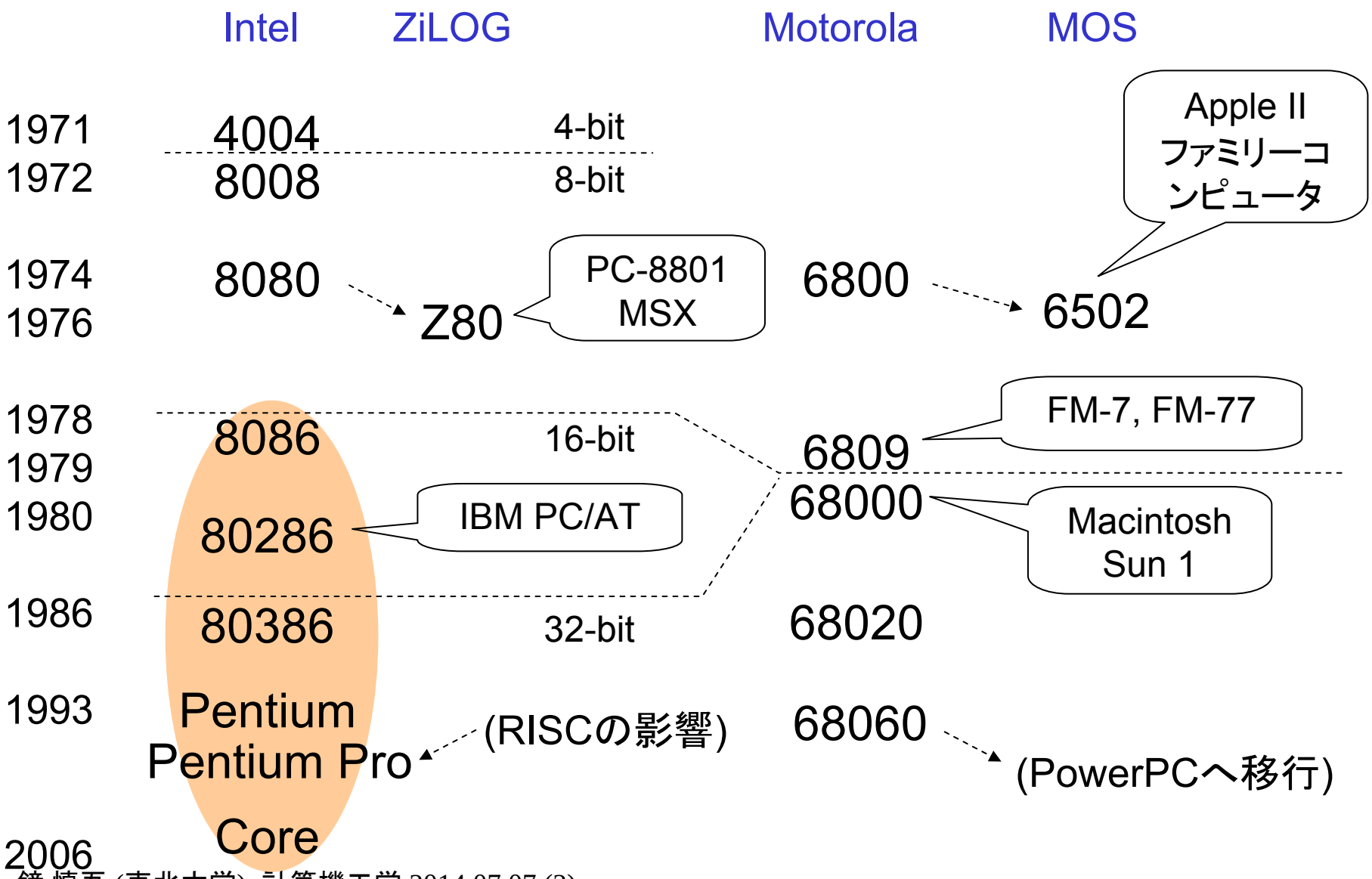
- 歴史的には，最初はすべて布線論理制御
- 命令大規模化・複雑化の流れによって，布線論理の設計コストが肥大化 → マイクロプログラム制御が主流になる (後にRISCの台頭によってその流れが変わる)

復習: 簡易版MIPSの制御回路

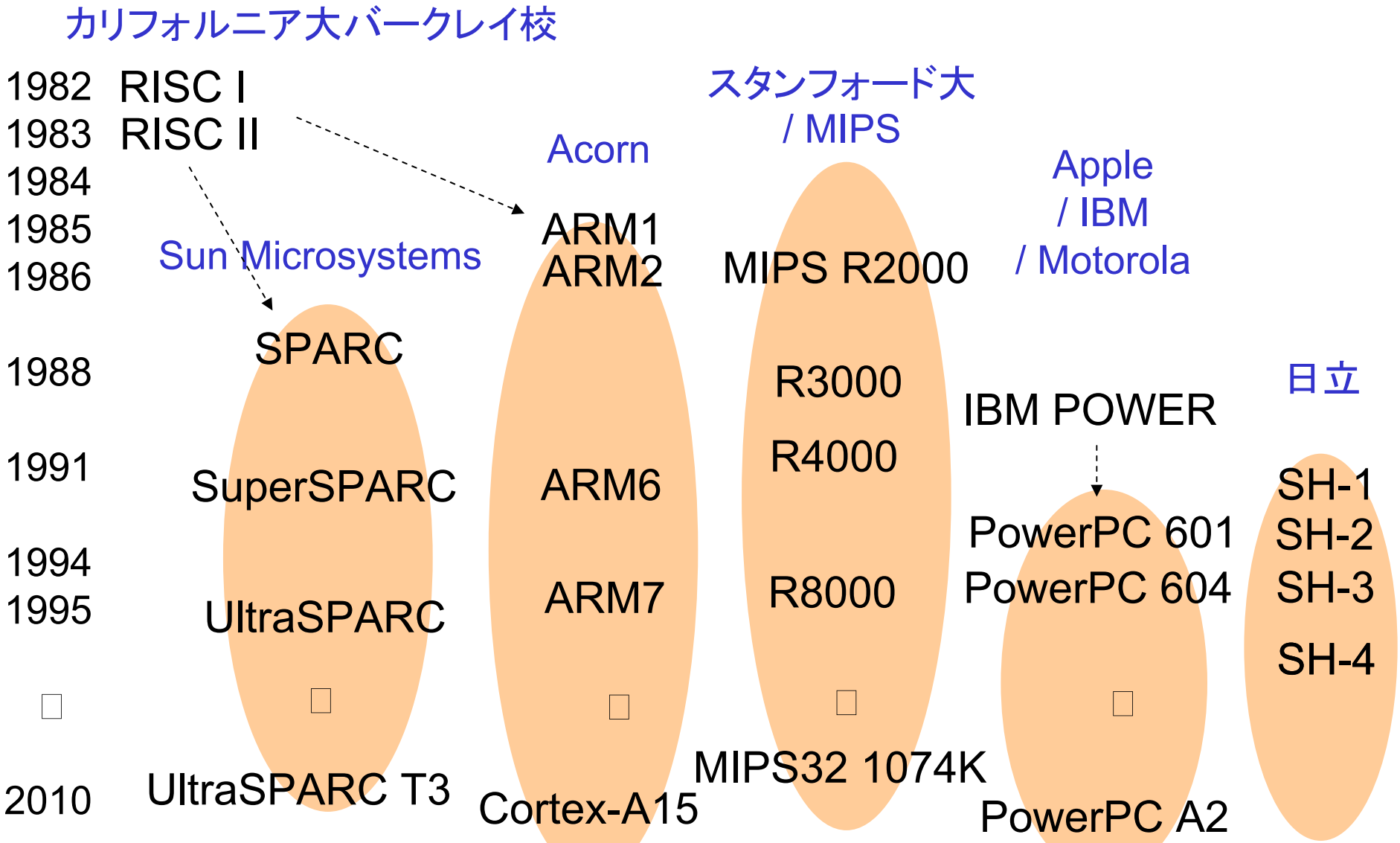


q_1	q_0	is_branch	q_1^{next}	q_0^{next}	IRen	DRen	GPRen
0	0	0	0	1	1	0	0
0	0	1	0	1	1	0	0
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	0
1	0	0	0	0	0	0	1
1	0	1	0	0	0	0	1
1	1	0	*	*	*	*	*
1	1	1	*	*	*	*	*

マイクロプロセッサの系譜 (CISC)



マイクロプロセッサの系譜 (RISC)



RISC型とCISC型

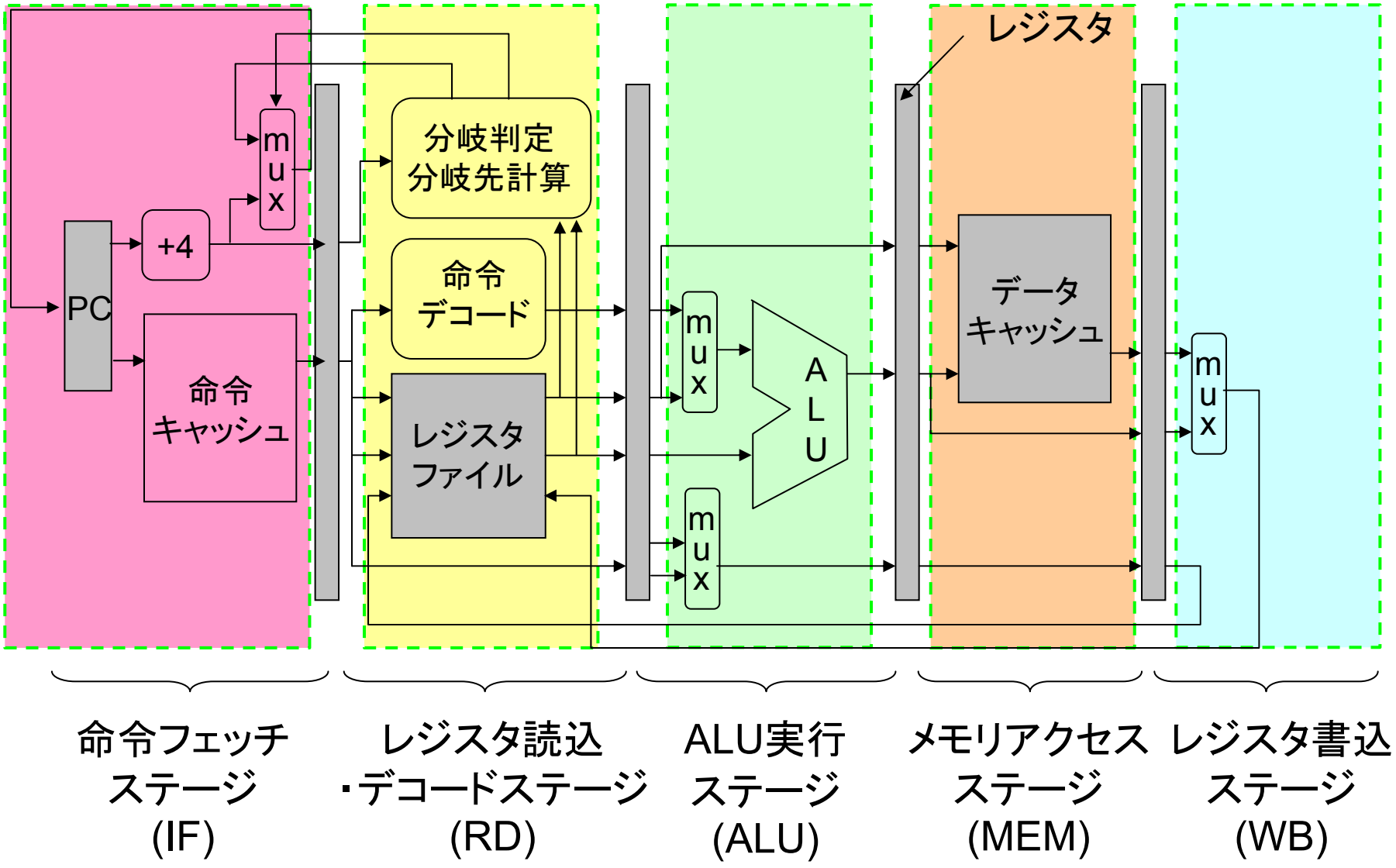
- キャッシュメモリの普及 (命令の高速読み出しが可能)
  - 実行命令数を減らす意味が薄れた
  - マイクロプログラム制御が足かせになってきた
- 高級言語の普及 (コンパイラは複雑な命令を使いこなせない)
- 回路自動設計の普及 (布線論理の設計が容易に)

→ RISC (Reduced Instruction Set Computer) 化の流れ
(cf. CISC: Complex Instruction Set Computer)

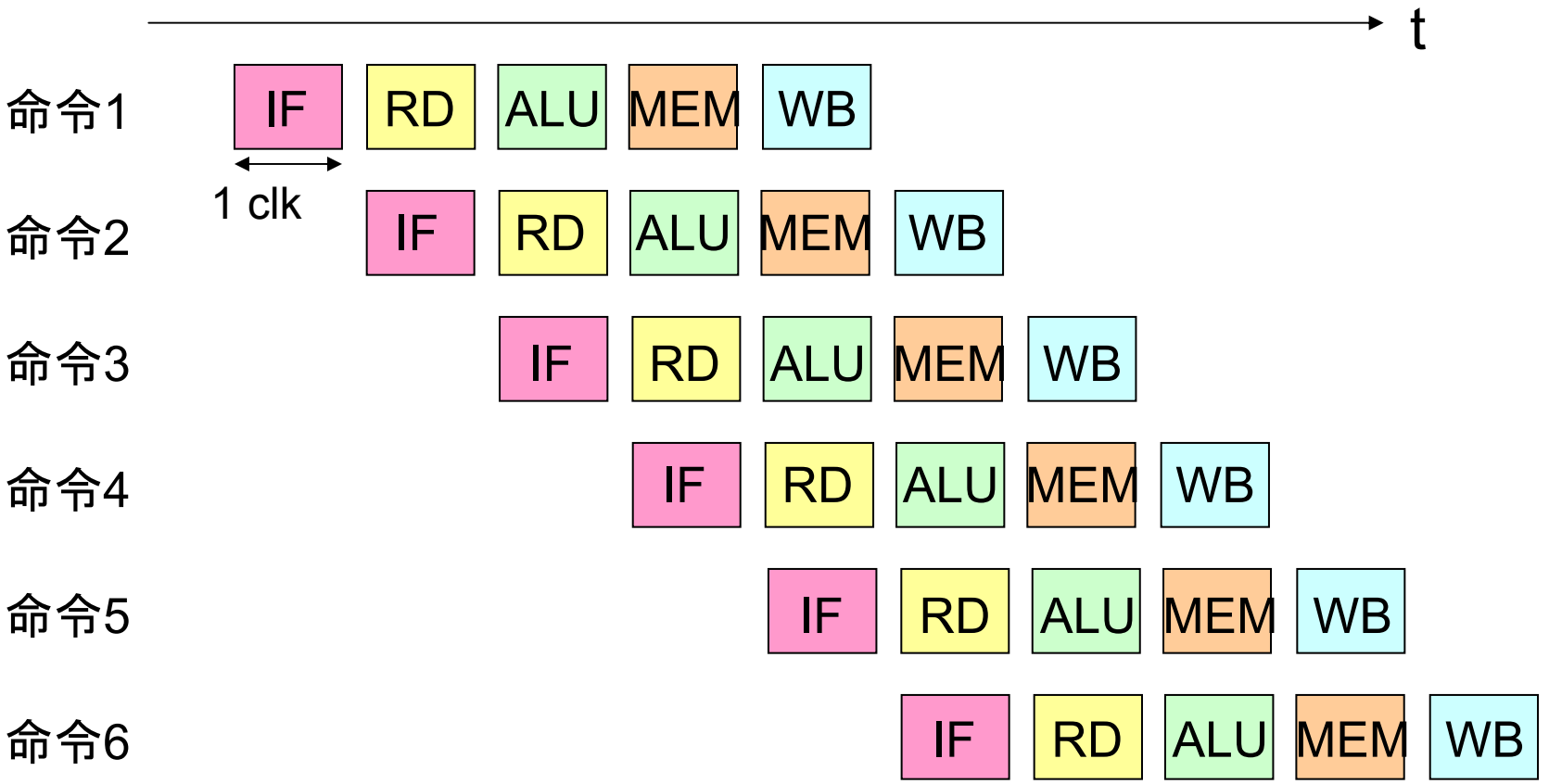
- 単純で規則的な命令セット
- 布線論理制御
- 実行命令数 増大，CPI 減少，クロックサイクル時間 減少
- パイプラインとの親和性

MIPS は最も典型的な RISC 型アーキテクチャ
最近は，Intel のアーキテクチャを除くほぼすべてがRISC
(Intel アーキテクチャも内部的にはほとんど RISC)

MIPSのパイプライン化

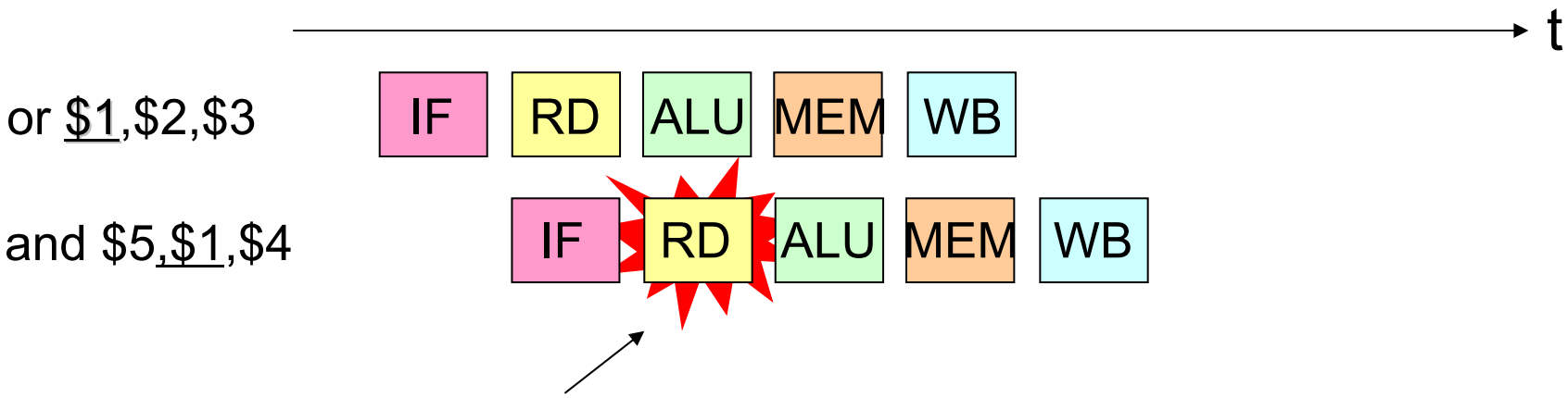


パイプラインによる命令実行



しかし，これで常に1命令1クロックで動けるようになるわけではなく，
パイプラインが正常に動作できない状況（ハザード）が存在する

データハザード

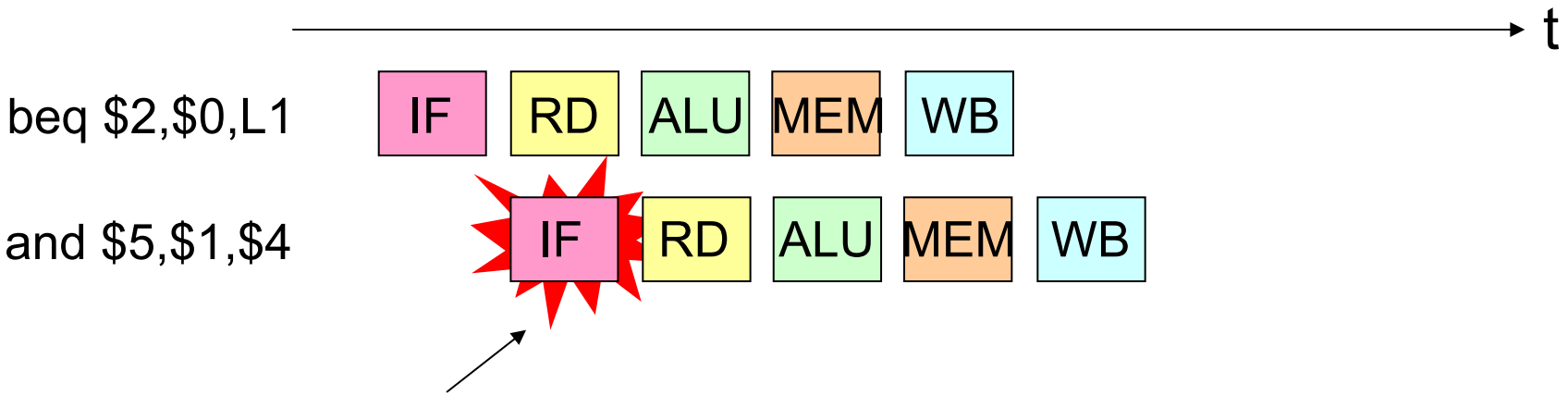


この時点では，前の命令の計算結果がまだ $1 に入っていない

命令間のデータ依存性によるハザード

例) 直前の命令の実行結果がレジスタ書き込まれる前に後続命令がそのレジスタを読み出してしまう

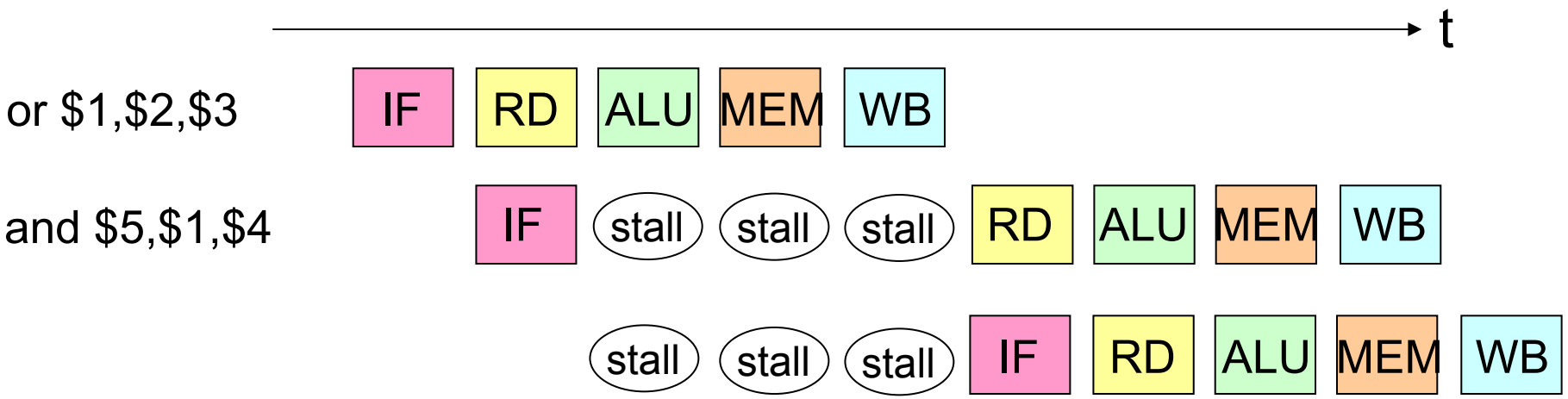
制御ハザード



命令の実行順序によるハザード

例) 直前の分岐命令による分岐先が定まらないうちに，次の命令の実行が始まってしまう

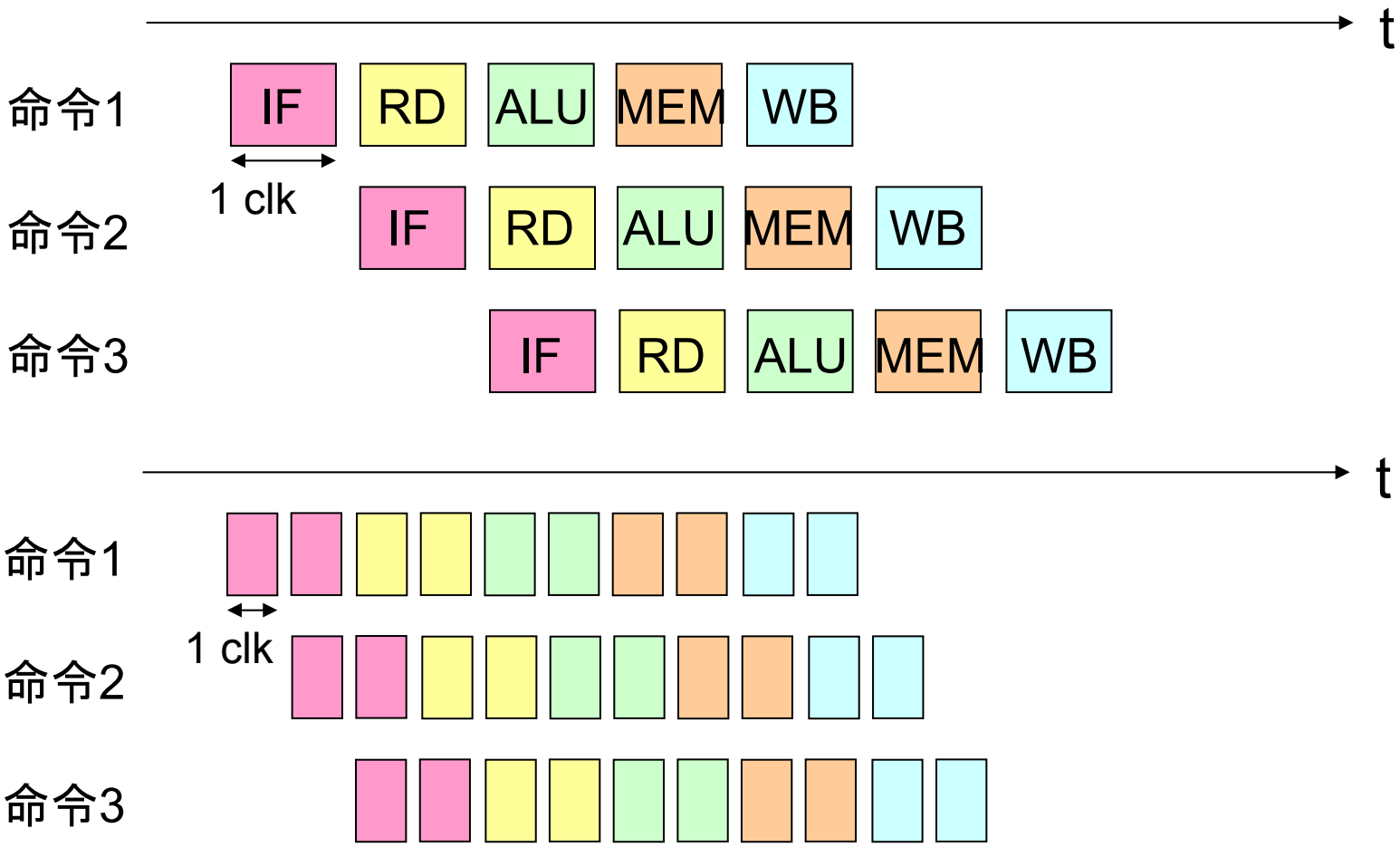
ハザードが発生した場合の動作: ストール



実行可能な状態になるまでパイプラインの動作を止める

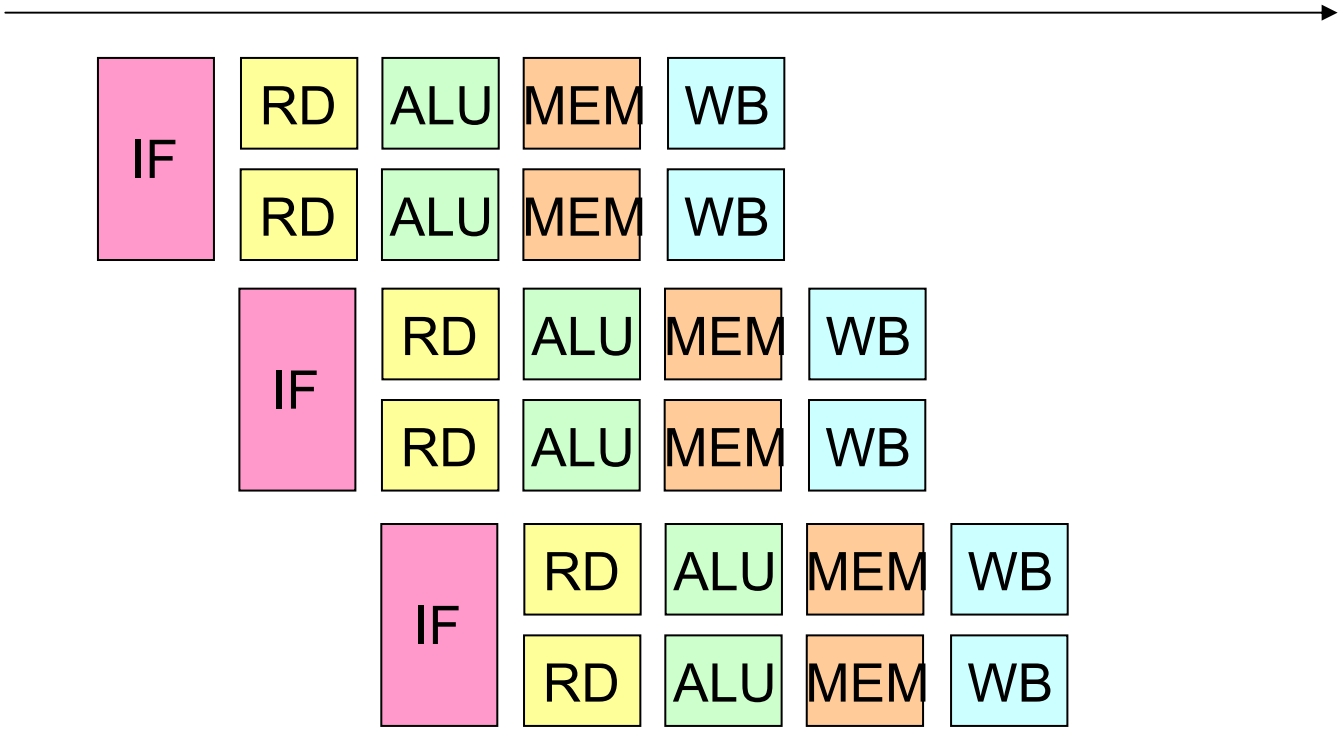
- どんなハザードにも適用可能だが，パフォーマンスが低下する
- ストールせずに済ますための対策がいろいろと講じられている
  - 例）データフォワーディング（レジスタを経由せずに後続命令へデータを渡す）
  - 例）遅延分岐・遅延ロード（分岐命令やロード命令の効果は1クロック遅れて現れることにして，命令スケジューリングをコンパイラに任せる）
  - 例）動的分岐予測

クロックサイクルのさらなる短縮



パイプラインをさらに細分化
(e.g. Pentium 4 は 20段)

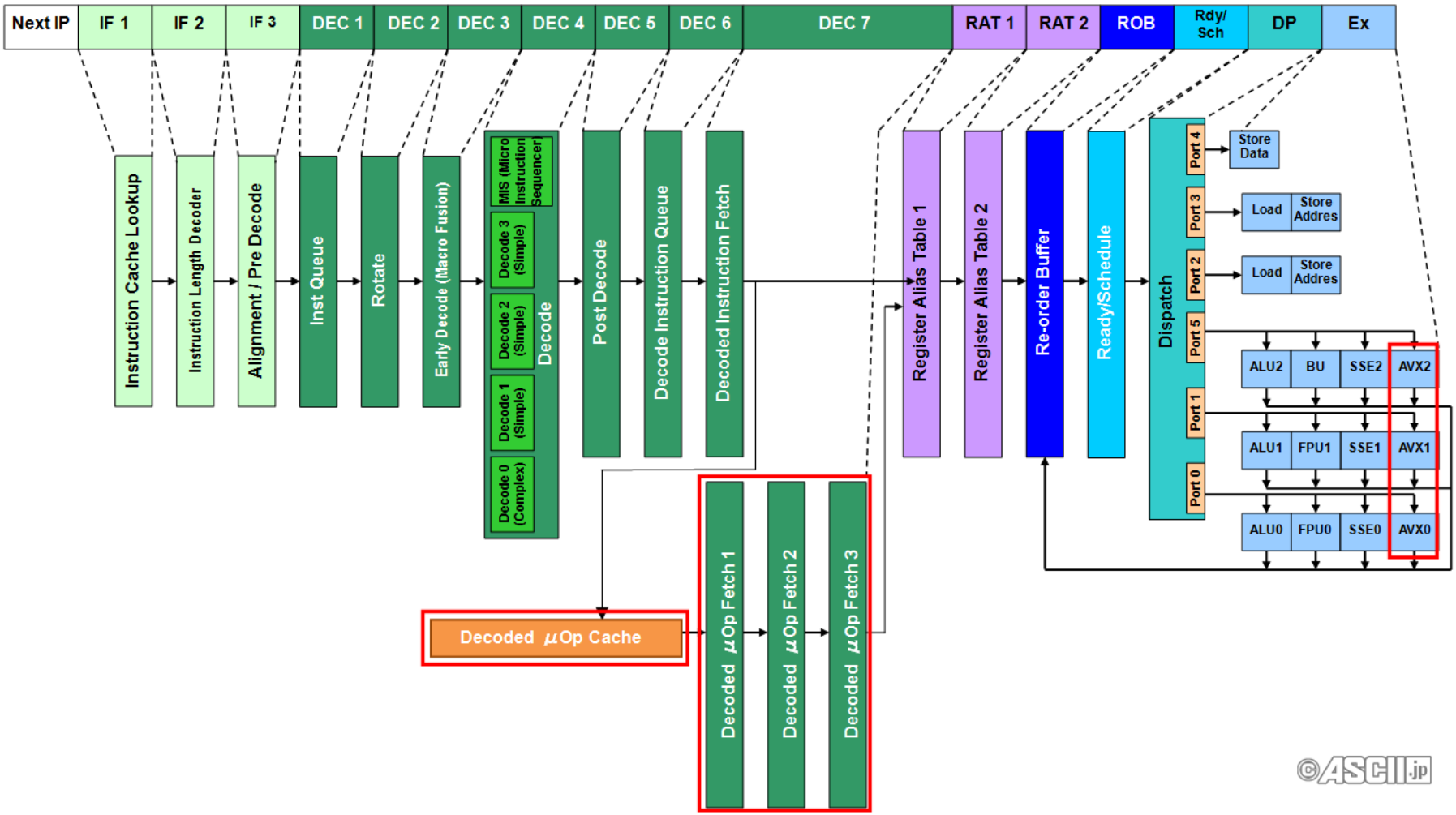
CPIのさらなる削減: 命令レベル並列性



スーパースカラ:

- 演算器を多重化し，複数命令を同時実行
- 同時実行可能かどうかはハードウェアが動的に判定

例: Core i7 Sandy Bridge



http://ascii.jp/elem/000/000/724/724498/

データ並列性

画像処理，音声処理，ある種の科学技術計算など
→ 同じ演算を多数のデータに適用することが多い
(SIMD; Single Instruction stream Multiple Data stream)

SIMD型並列処理の実現形態:

空間並列: 同じ演算器を多数並べる

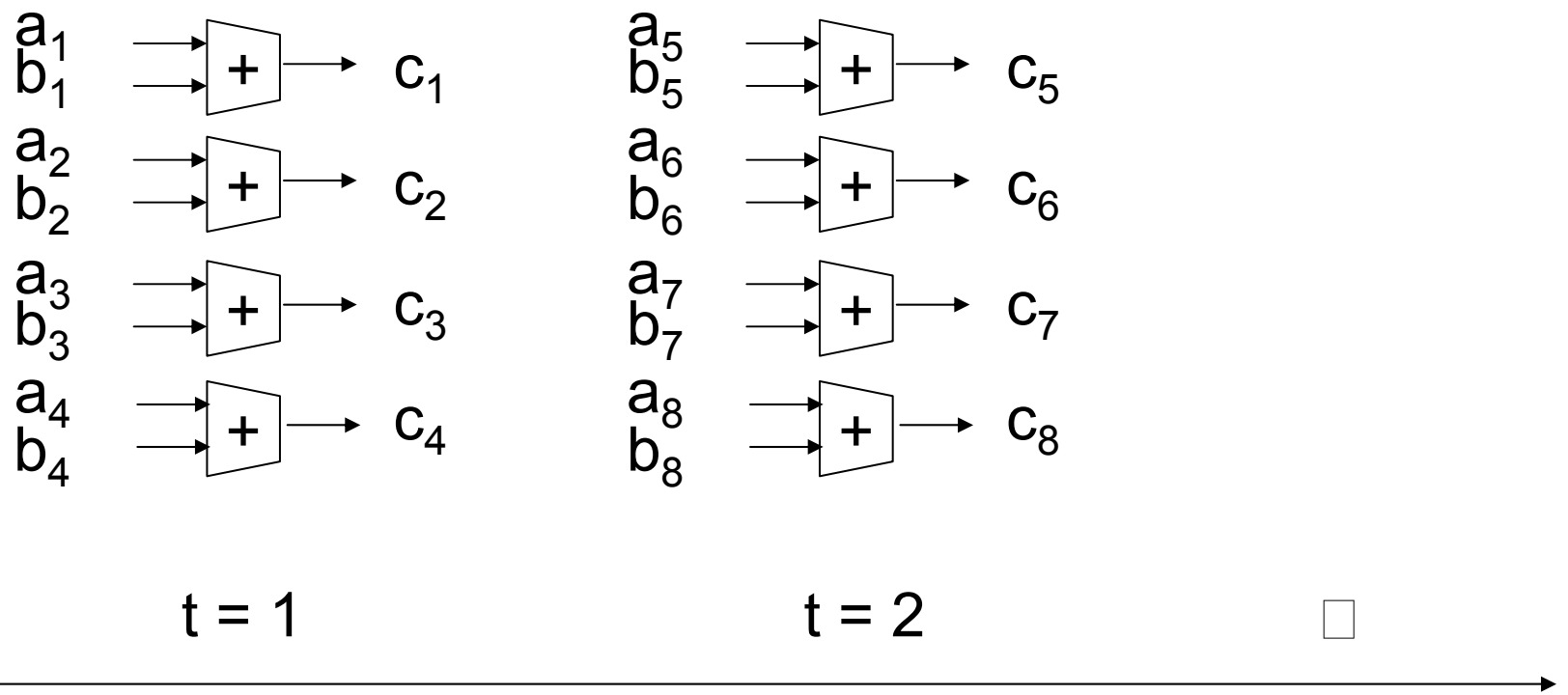
例) マルチメディア命令 (MMX, SSE 命令など)

例) GPU (Graphic Processing Unit)

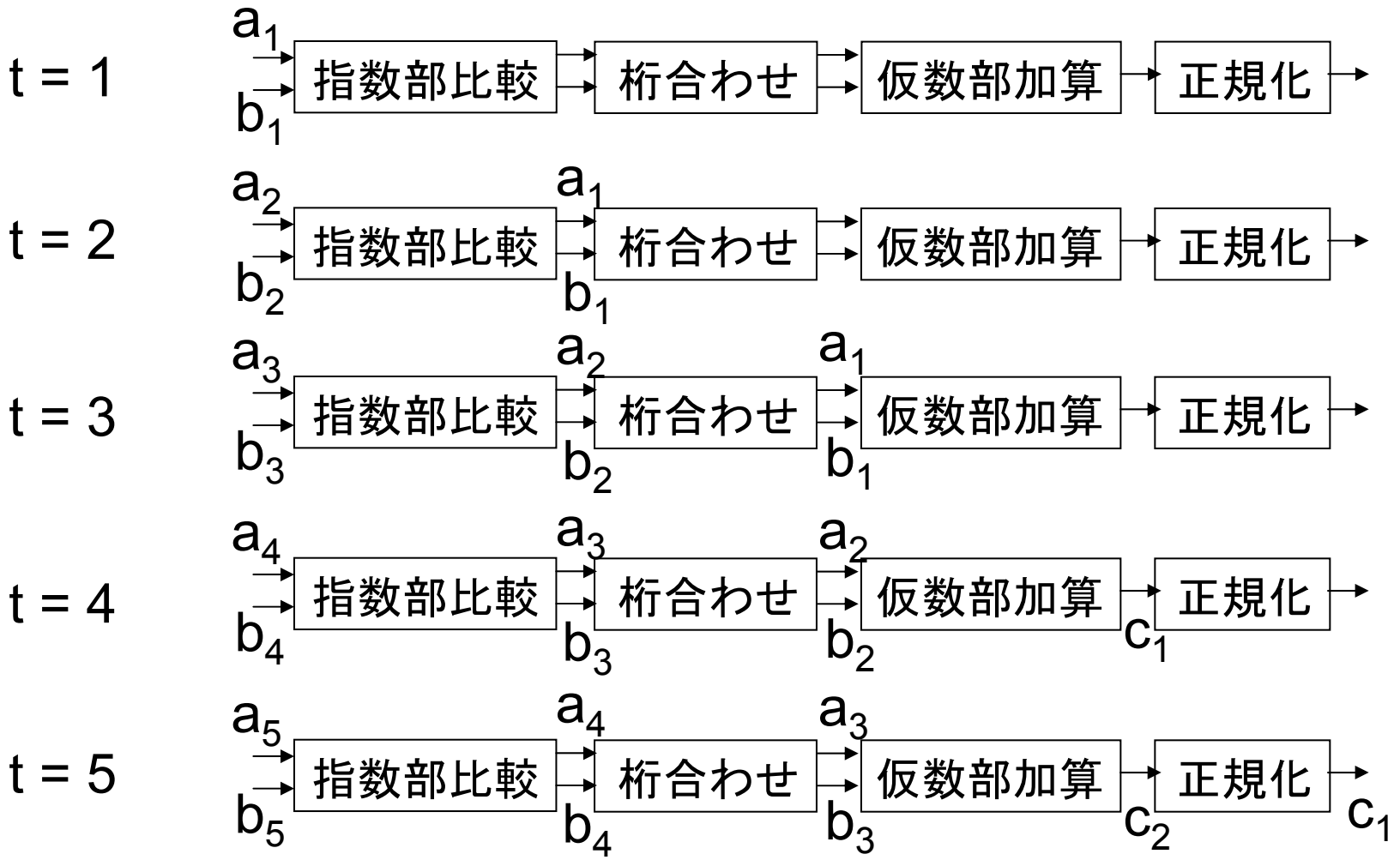
時間並列: 処理を複数のステージに分割してパイプライン化
（ベクトル演算と呼ばれる場合がある）

空間並列の例

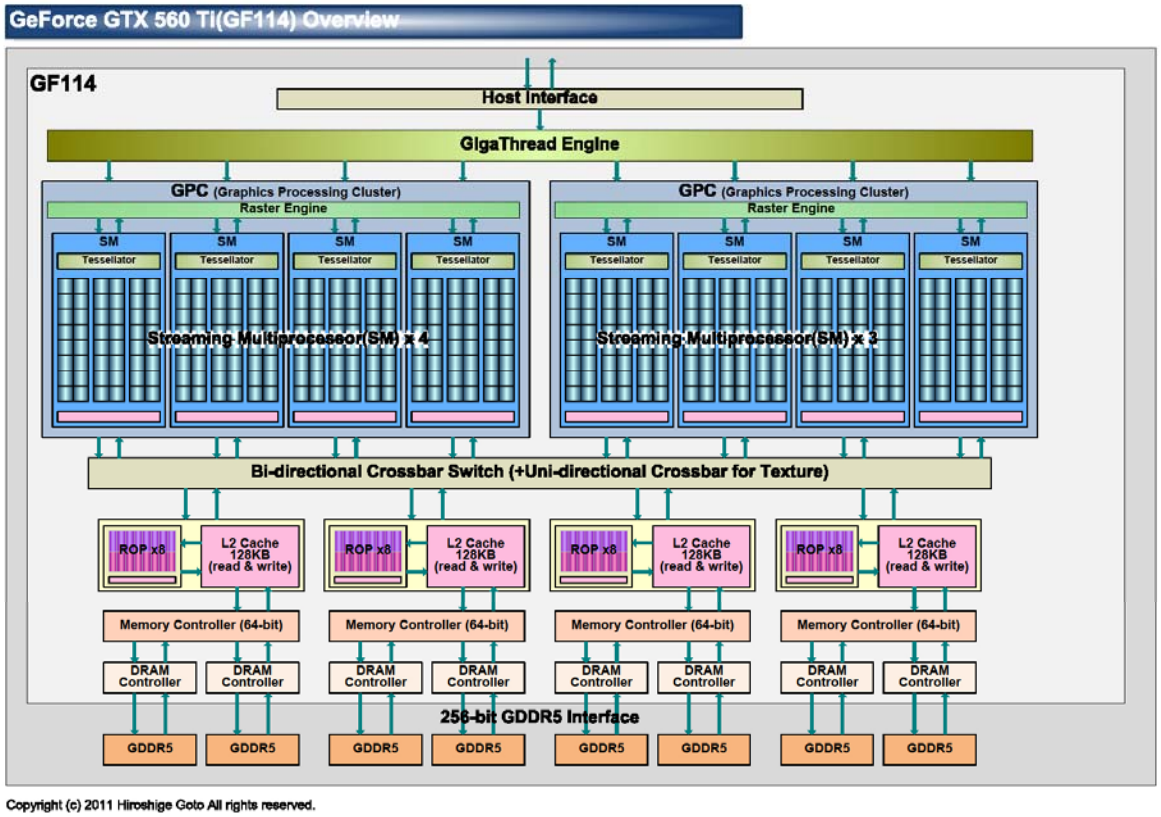
浮動小数点ベクトル $[a_1, a_2, \cdots, a_n]$ と $[b_1, b_2, \cdots, b_n]$ の加算



時間並列の例


t = 1
t = 2
t = 3
t = 4
t = 5
指数部比較
桁合わせ
仮数部加算
正規化
a1
b1
a2
b2
a3
b3
a4
b4
a5
b5
c1
c2

例: GPU (Graphics Processing Unit)



NVIDIA GeForce GTX 560 Ti

http://pc.watch.impress.co.jp/docs/column/kaigai/20110126_422573.html

http://www.nvidia.co.jp/object/product-geforce-gtx-560ti-jp.html

スレッドレベル並列性

- クロックサイクル時間短縮: 消費電力の限界
- 命令レベル並列性: 3程度が限界
- データ並列性: アプリケーション依存

→ スレッドレベル並列性の活用へ

複数のプログラム，あるいはプログラム内の複数の処理の流れ（thread of control）からであれば，同時に実行できる命令を容易に取り出すことができる

- 同時マルチスレッディング: スーパースカラプロセッサにおいて，複数のスレッドからの命令を取り出して実行
  例) Intel の Hyper-Threading Technology
- マルチコア: 複数のプロセッサをチップ上に集積

Intel Core i7 (2008)