マイクロアーキテクチャ攻撃 2

九州大学　サイバーセキュリティセンター
谷本 輝夫

マイクロアーキテクチャ攻撃 2

午前（10:00 - 12:00）

- 前回のおさらい（マイクロアーキテクチャ攻撃とは）
- メモリに対する攻撃
  - Rowhammer 攻撃
- マイクロアーキテクチャ攻撃の影響と対応
- 前回のおさらい（Spectre）

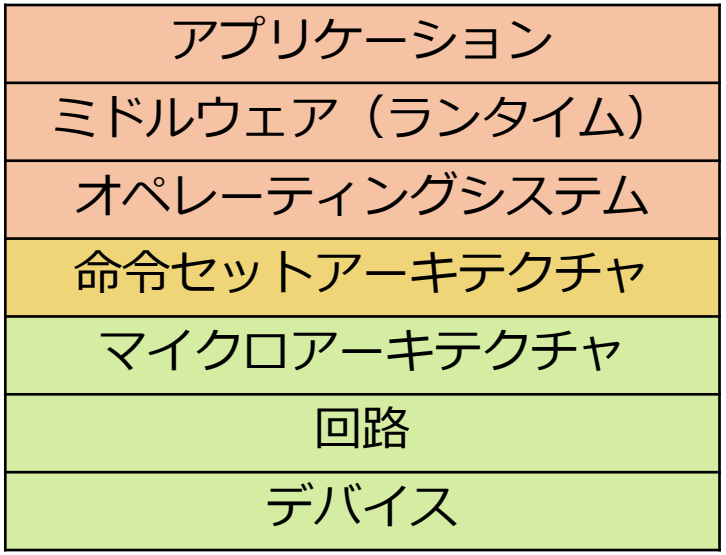
午後（13:00 – 14:00）

- 実習（攻撃の再現実験）
  - プロセッサシミュレータを使った Spectre の解析

前回のおさらい（マイクロアーキテクチャ攻撃とは）

マイクロアーキテクチャとは

- コンピュータ・システムの抽象化レイヤの１つ
  - 命令セットアーキテクチャ（ISA）の実装におけるアーキテクチャを指す
  - ソフトウェア／ハードウェアの境界に位置



マイクロアーキテクチャ攻撃とは

- マイクロアーキテクチャの実装に起因する脆弱性に対する攻撃
- ソフトウェアベース・マイクロアーキテクチャ攻撃
  - ハードウェア・ソフトウェア間で本来規定されている機能（入力→処理→出力）以外の副作用を起こすことによる攻撃が多数報告されている
  - ソフトウェアから見えるコンピュータシステムの状態（アーキテクチャ・ステート）とハードウェアの内部状態（マイクロアーキテクチャ・ステート）の違いに着目し、サイドチャネル攻撃により本来取得できない情報を取得・推測する手法
    - Meltdown [M. Lipp+ 2018]
    - Spectre [P. Kocher+ 2018]
    - …

    } CPUに対する攻撃
  - 命令実行によりアーキテクチャ・ステートを書き換える手法
    - Rowhammer [Y. Kim+ 2014]

[M. Lipp+ 2018] Meltdown: Reading Kernel Memory from User Space
[P. Kocher+ 2018] Spectre Attacks: Exploiting Speculative Execution
[Y. Kim+ 2014] Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors

マイクロアーキテクチャ攻撃とは

- マイクロアーキテクチャの実装に起因する脆弱性に対する攻撃
- ソフトウェアベース・マイクロアーキテクチャ攻撃
  - ハードウェア・ソフトウェア間で本来規定されている機能（入力→処理→出力）以外の副作用を起こすことによる攻撃が多数報告されている

前回紹介

  - ソフトウェアから見えるコンピュータシステムの状態（アーキテクチャ・ステート）とハードウェアの内部状態（マイクロアーキテクチャ・ステート）の違いに着目し、サイドチャネル攻撃により本来取得できない情報を取得・推測する手法
    - Meltdown [M. Lipp+ 2018]
    - Spectre [P. Kocher+ 2018]
    - …

    } CPUに対する攻撃

  - 命令実行によりアーキテクチャ・ステートを書き換える手法
    - Rowhammer [Y. Kim+ 2014]

[M. Lipp+ 2018] Meltdown: Reading Kernel Memory from User Space
[P. Kocher+ 2018] Spectre Attacks: Exploiting Speculative Execution
[Y. Kim+ 2014] Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors

マイクロアーキテクチャ攻撃とは

- マイクロアーキテクチャの実装に起因する脆弱性に対する攻撃
- ソフトウェアベース・マイクロアーキテクチャ攻撃
  - ハードウェア・ソフトウェア間で本来規定されている機能（入力→処理→出力）以外の副作用を起こすことによる攻撃が多数報告されている
  - ソフトウェアから見えるコンピュータシステムの状態（アーキテクチャ・ステート）とハードウェアの内部状態（マイクロアーキテクチャ・ステート）の違いに着目し、サイドチャネル攻撃により本来取得できない情報を取得・推測する手法
    - Meltdown [M. Lipp+ 2018]
    - Spectre [P. Kocher+ 2018]

    } CPUに対する攻撃
    - …

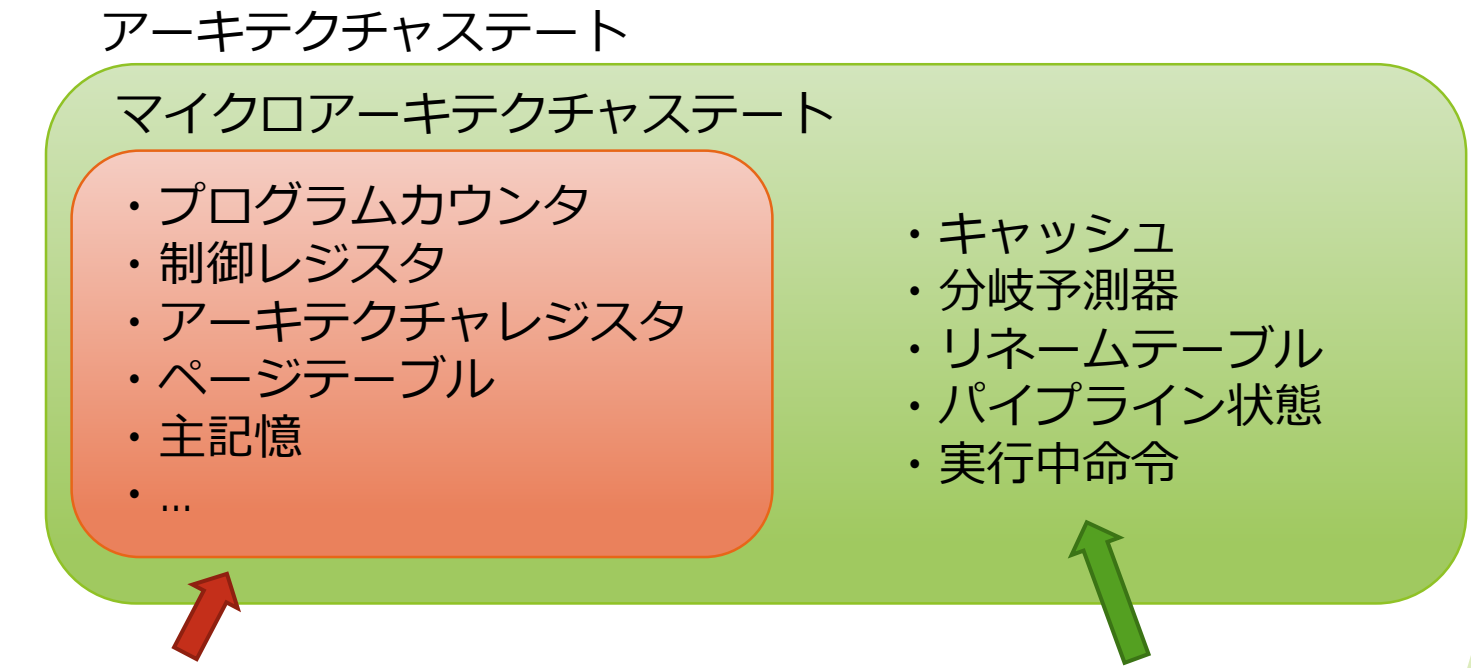
今回紹介

- 命令実行によりアーキテクチャ・ステートを書き換える手法
  - Rowhammer [Y. Kim+ 2014]

[M. Lipp+ 2018] Meltdown: Reading Kernel Memory from User Space
[P. Kocher+ 2018] Spectre Attacks: Exploiting Speculative Execution
[Y. Kim+ 2014] Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors

主記憶に対する攻撃

アーキテクチャステートとマイクロアーキテクチャステートの関係



理解するために必要な知識

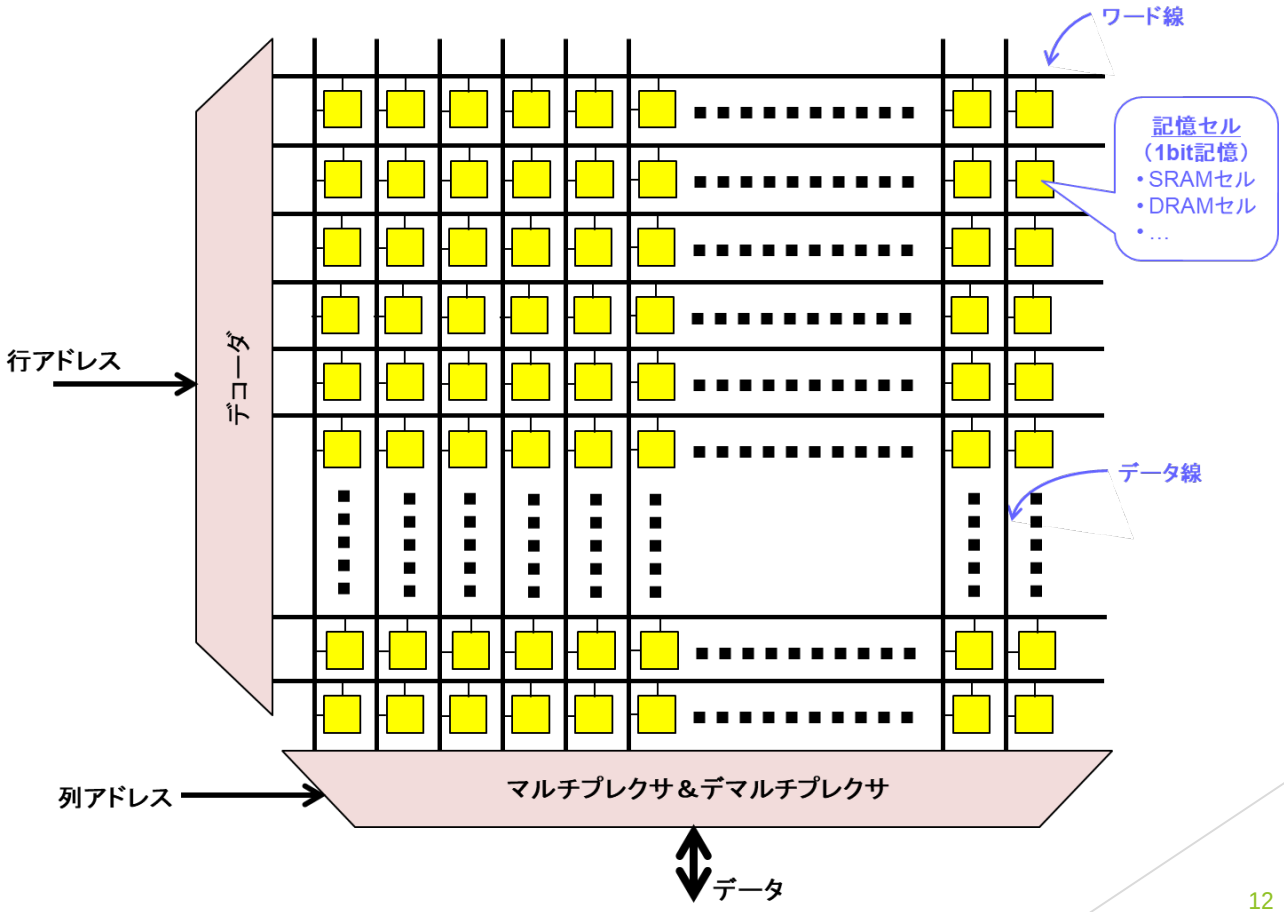
- 半導体記憶素子
- 半導体記憶素子の構造
- 半導体記憶素子へのアクセス
- 半導体記憶素子の記憶セル
  - DRAMセルの記憶メカニズム
- DRAMのリフレッシュ

半導体記憶素子

- Static Random Access Memory (SRAM)
  - 揮発性：電源断でデータ消失
  - 高速アクセス
- Dynamic Random Access Memory (DRAM)
  - 揮発性：電源断でデータ消失
  - （SRAMに比べて）大容量かつ安価
  - ダイナミック：リフレッシュ（記憶保持動作）が必要
- 他にもある
- 主記憶
  - アクセス速度、容量、コストのバランスからDRAMを利用

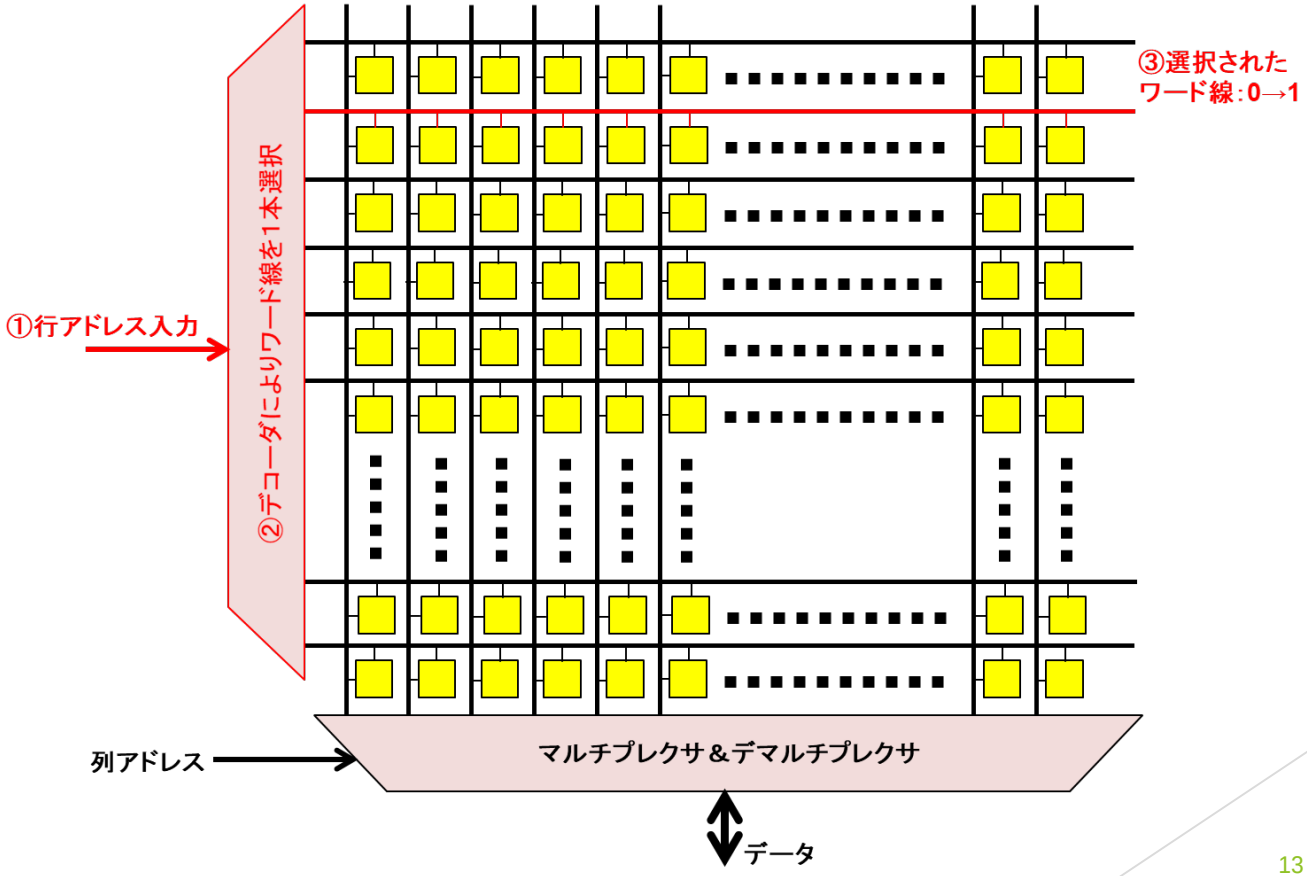
半導体記憶素子の構造

- 基本構造



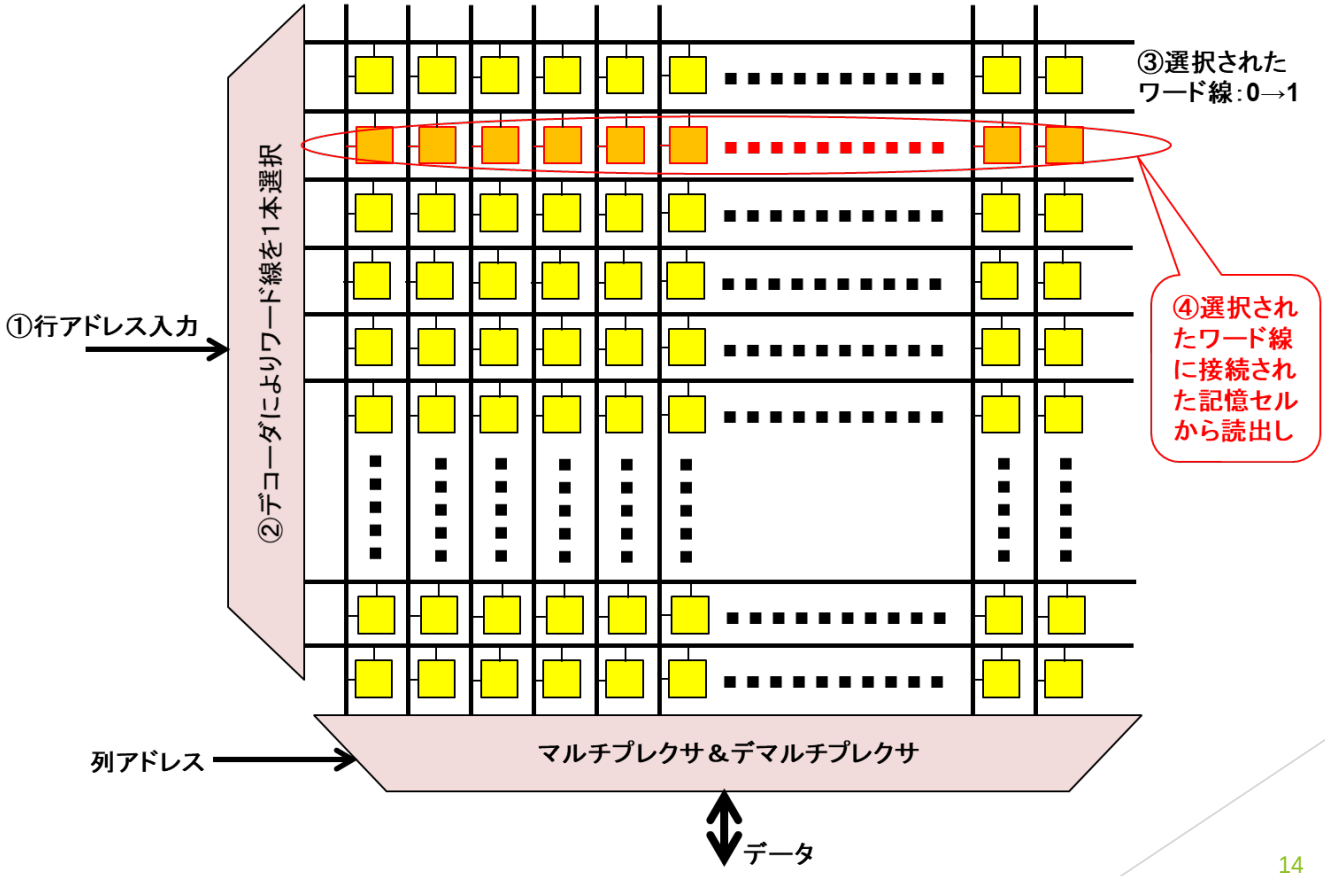
半導体記憶素子へのアクセス

- 動作１：行選択



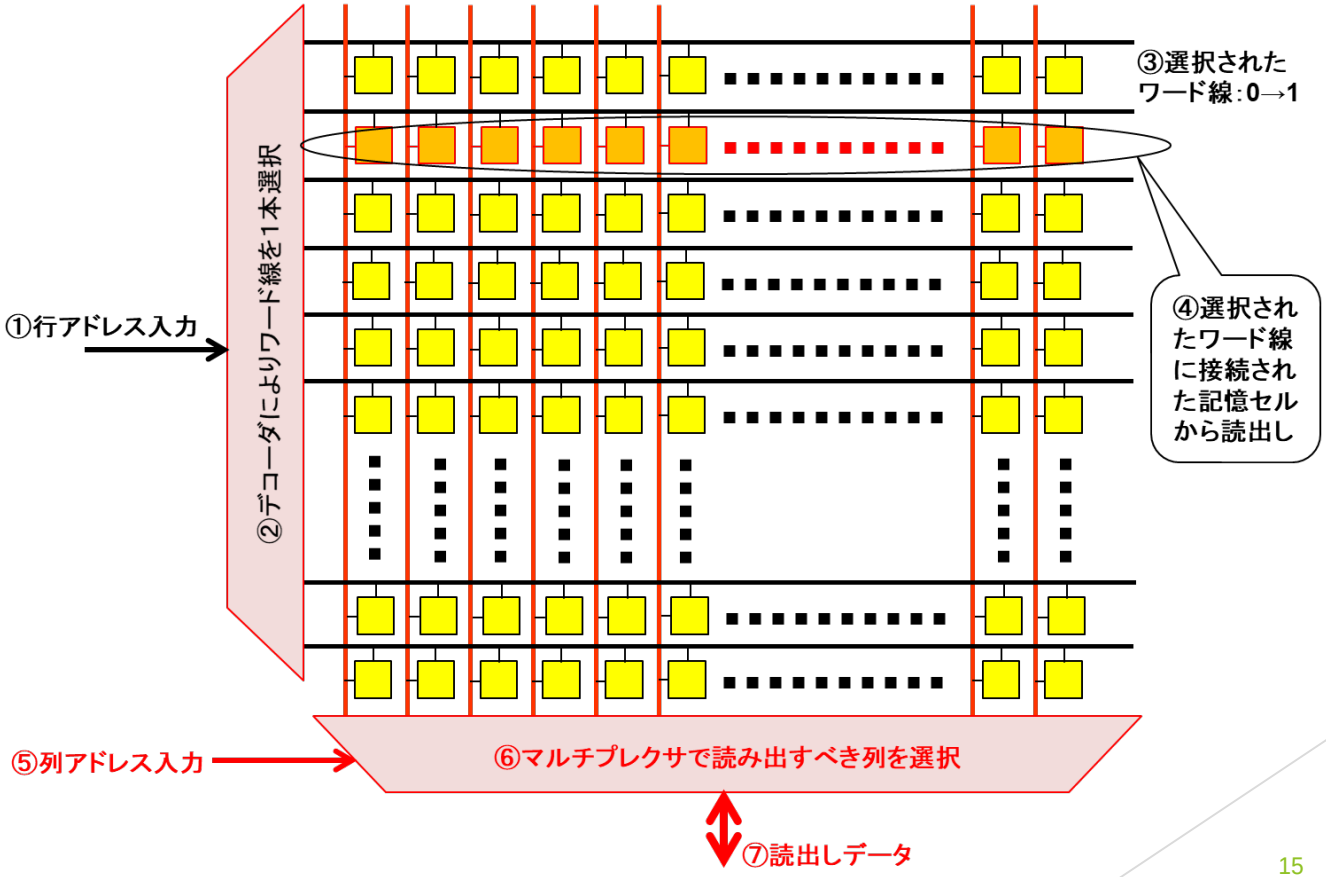
半導体記憶素子へのアクセス

- 動作２：記憶セル



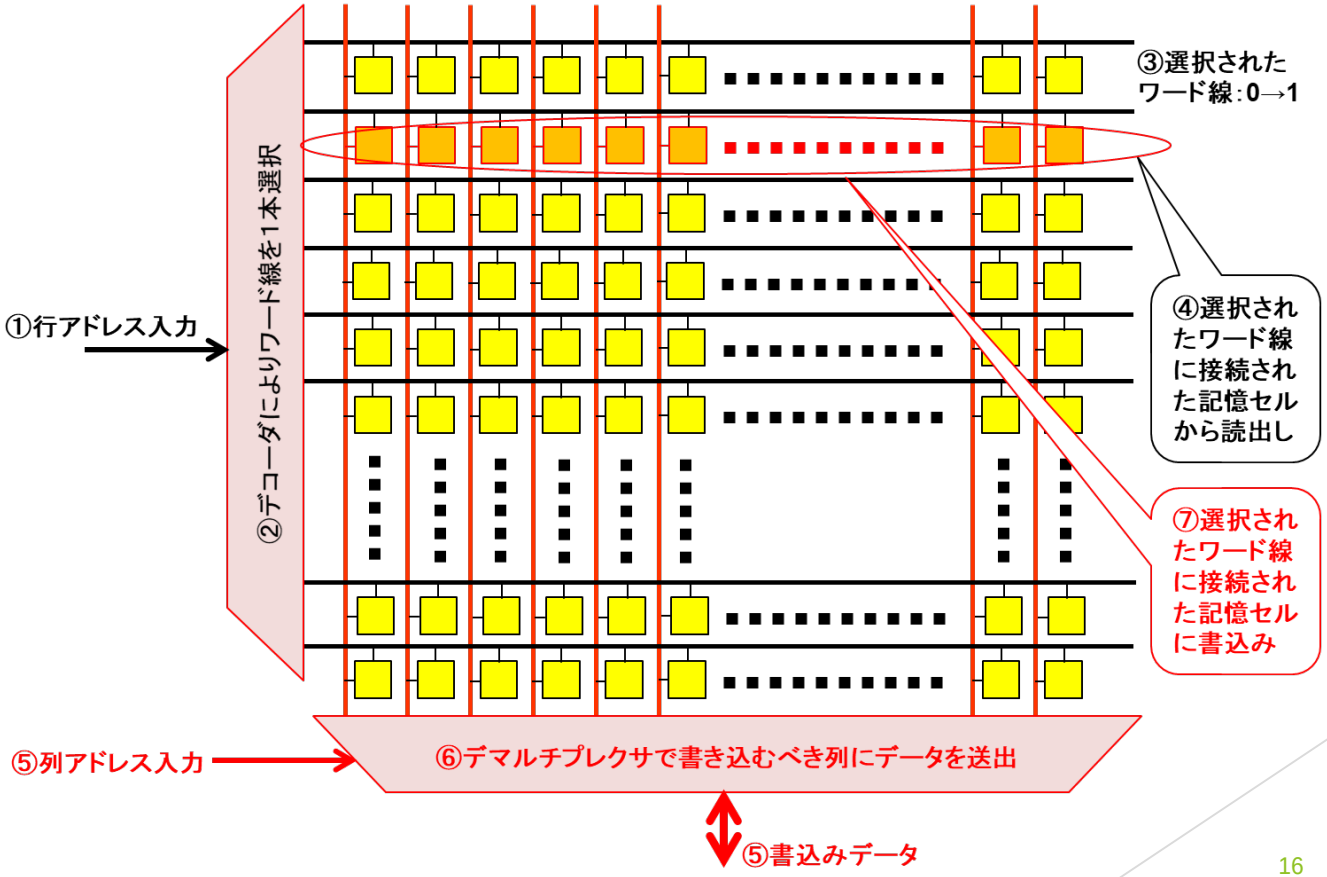
半導体記憶素子へのアクセス

- 動作３：列選択（読み出し）

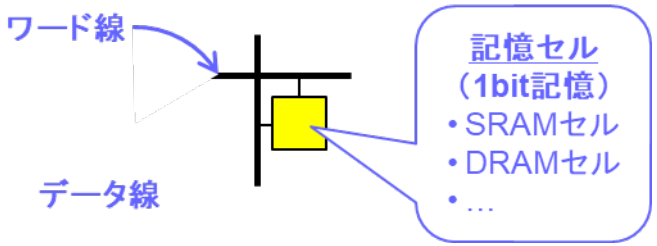


半導体記憶素子へのアクセス

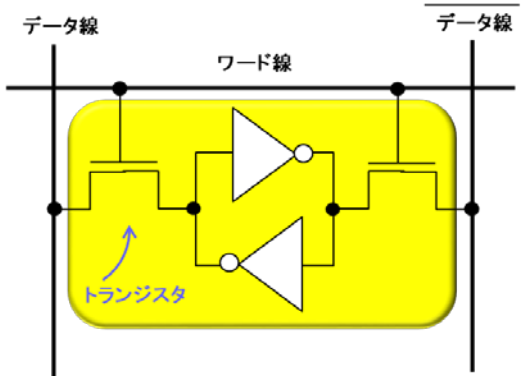
- 動作３：列選択（書き込み）



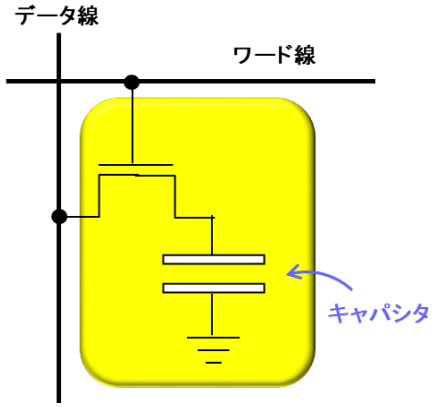
半導体記憶素子の記憶セル



- SRAMセル



- DRAMセル

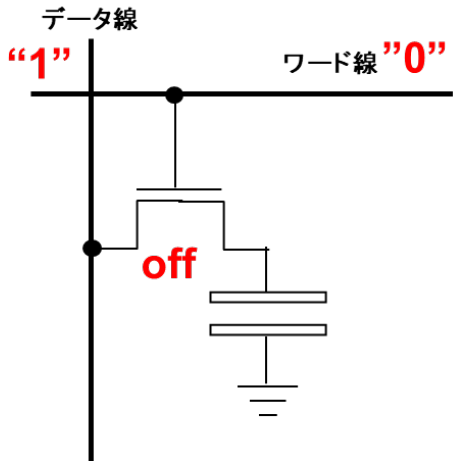


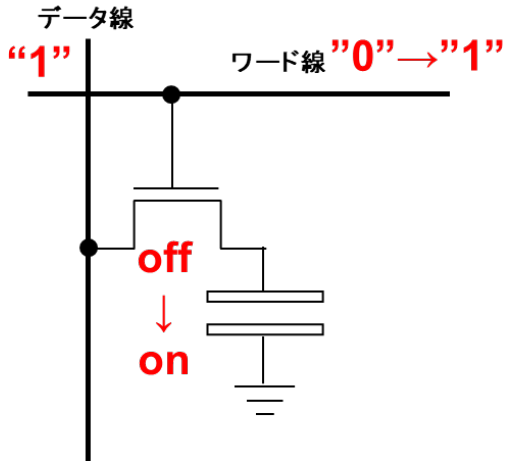
DRAMセルの記憶メカニズム

- データ”1”書き込み

① データ線にデータ”1”を印加

② ワード線を”0”から”1”へ（トランジスタをoffからonへ）

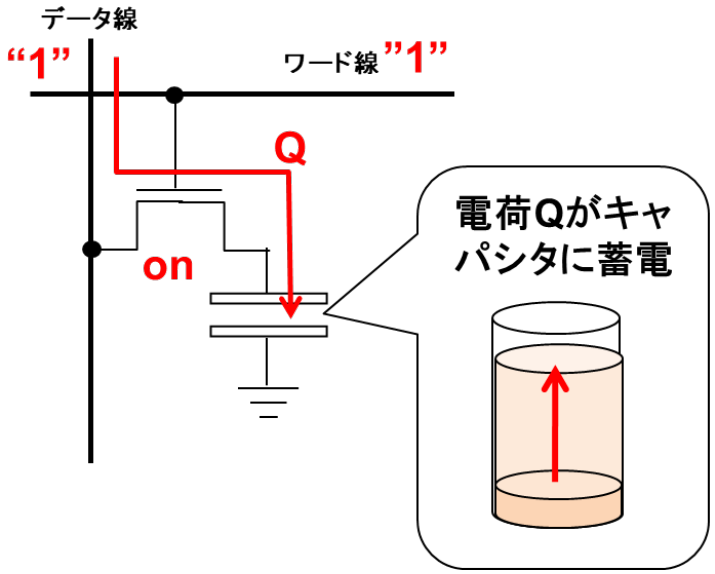




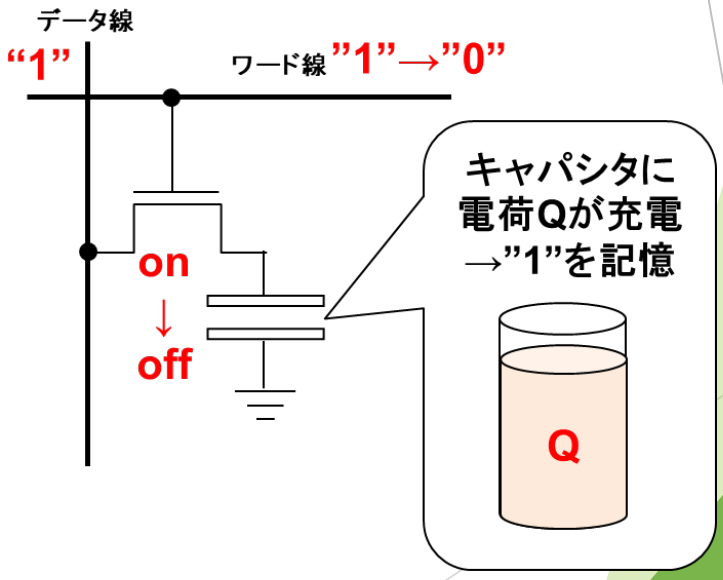
DRAMセルの記憶メカニズム

- データ”1”書き込み

③ データ線からキャパシタに電荷Qが流入



④ ワード線を”1”から”0”へ（トランジスタをonからoffへ）

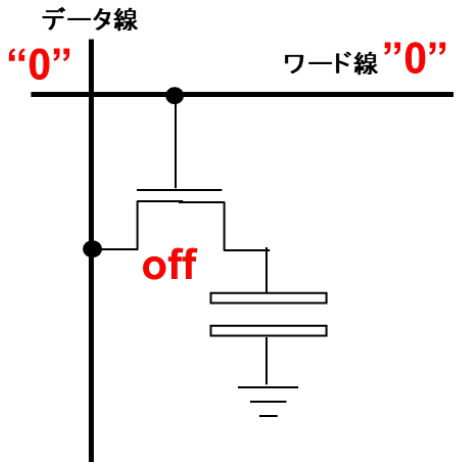


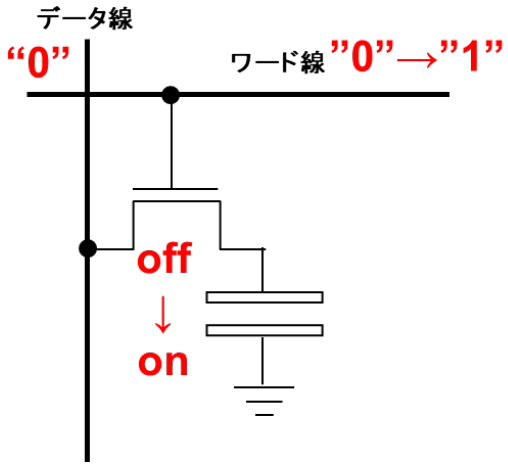
DRAMセルの記憶メカニズム

- データ”0”書き込み

① データ線にデータ”0”を印加

② ワード線を”0”から”1”へ（トランジスタをoffからonへ）

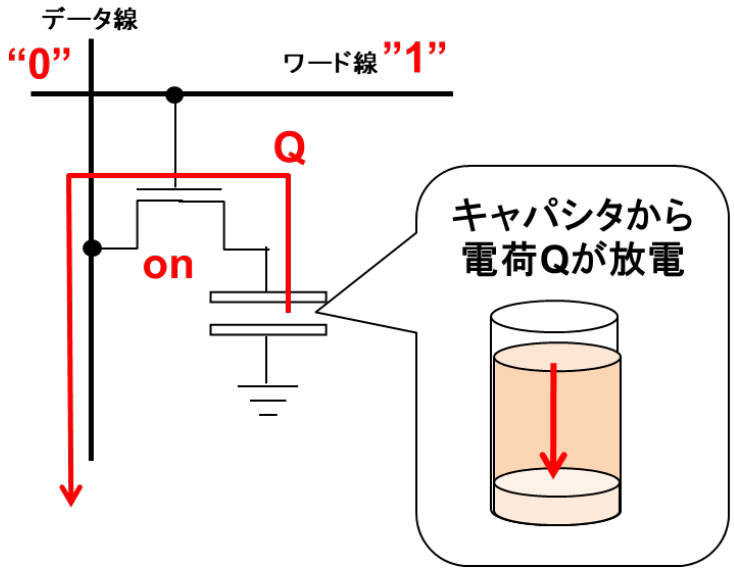




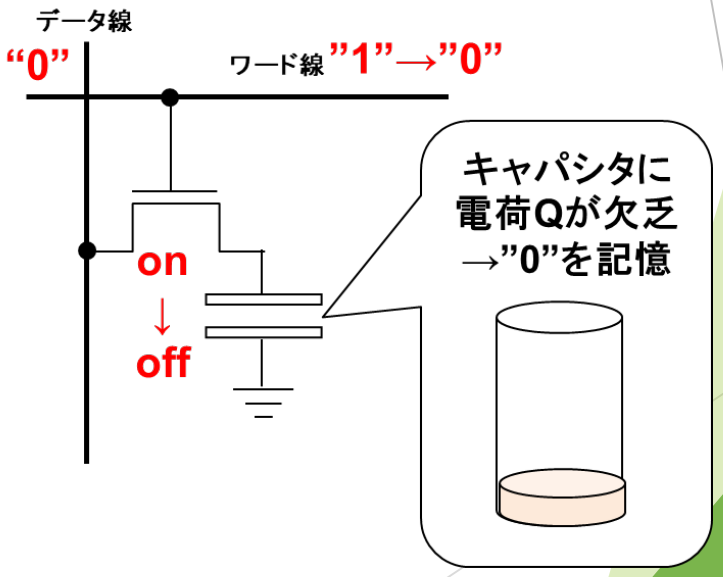
DRAMセルの記憶メカニズム

- データ”0”書き込み

③ キャパシタからデータ線に電荷Qが流出



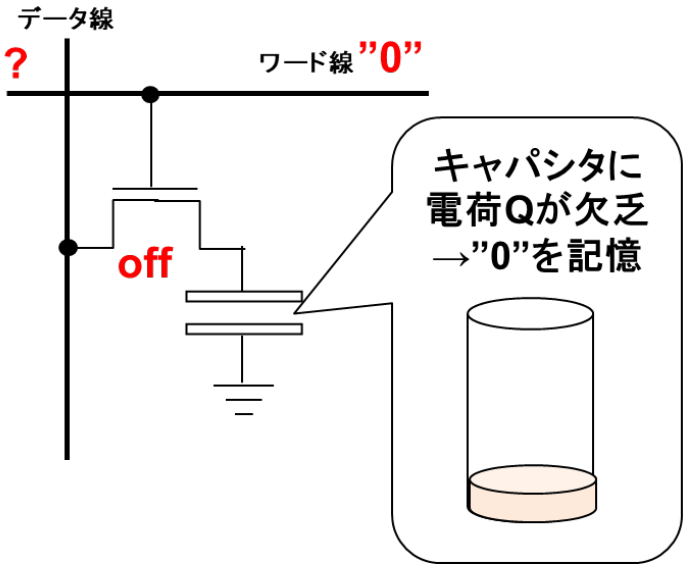
④ ワード線を”1”から”0”へ（トランジスタをonからoffへ）



DRAMセルの記憶メカニズム

- データ”0”書き込み

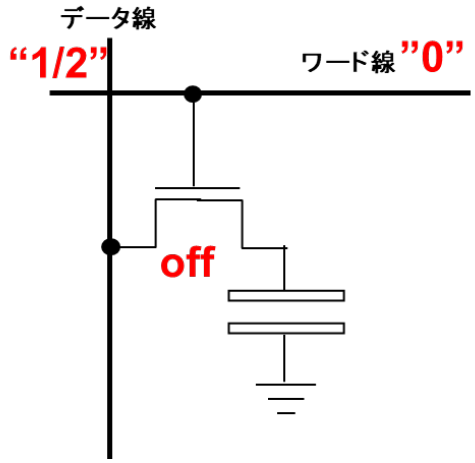
⑤ データ線からデータ”0”
を消去。記憶を保持



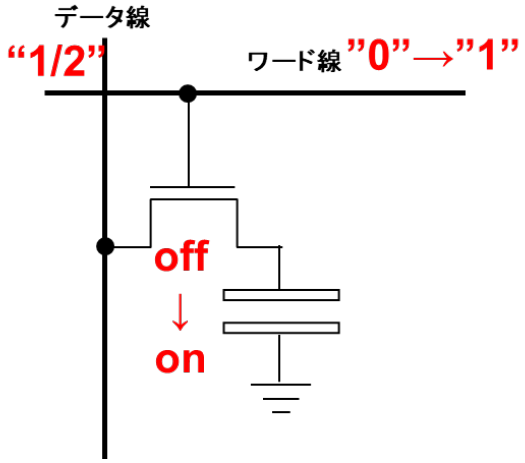
DRAMセルの記憶メカニズム

- データ”1”読み出し

① データ線にデータ”1/2”（”0”と”1”の中間値）を印加



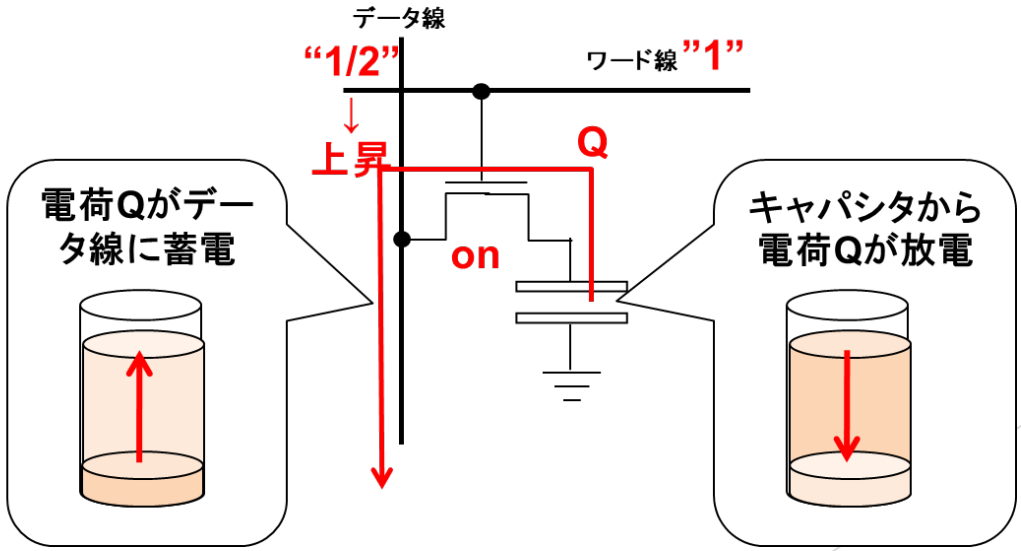
② ワード線を”0”から”1”へ（トランジスタをoffからonへ）



DRAMセルの記憶メカニズム

- データ”1”読み出し

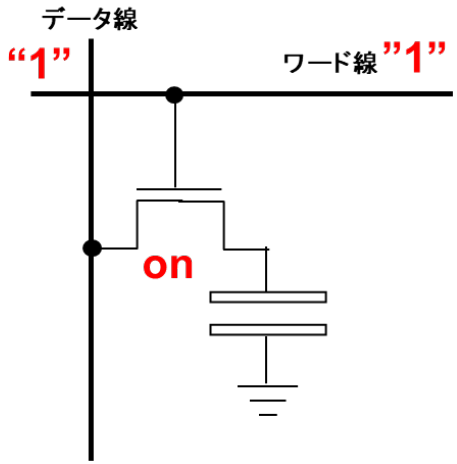
③ キャパシタからデータ線に電荷Qが流出
→データ線の電圧が上昇
→データ”1”を記憶していたと判断



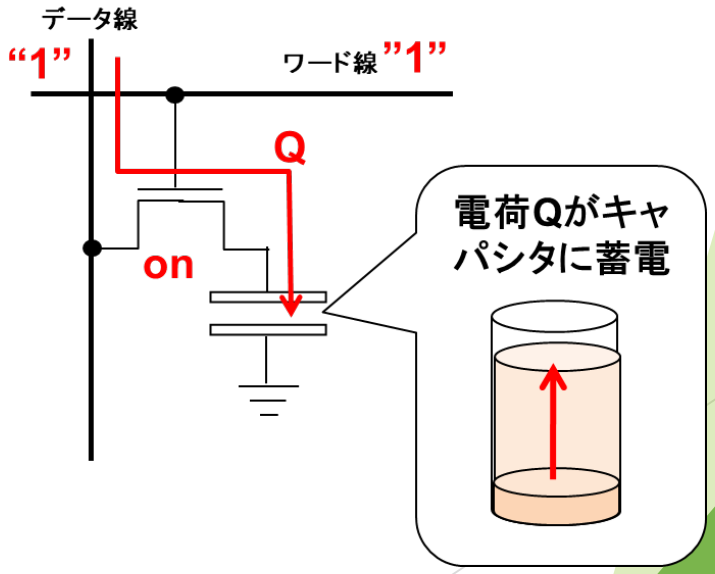
DRAMセルの記憶メカニズム

- データ”1”読み出し

④ 再びデータ”1”を書き込むために、データ線にデータ”1”を印加



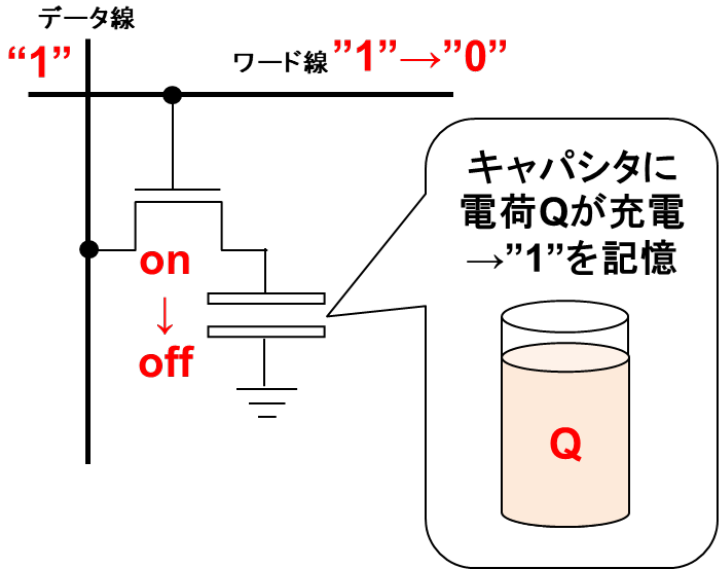
⑤ データ線からキャパシタに電荷Qが流入



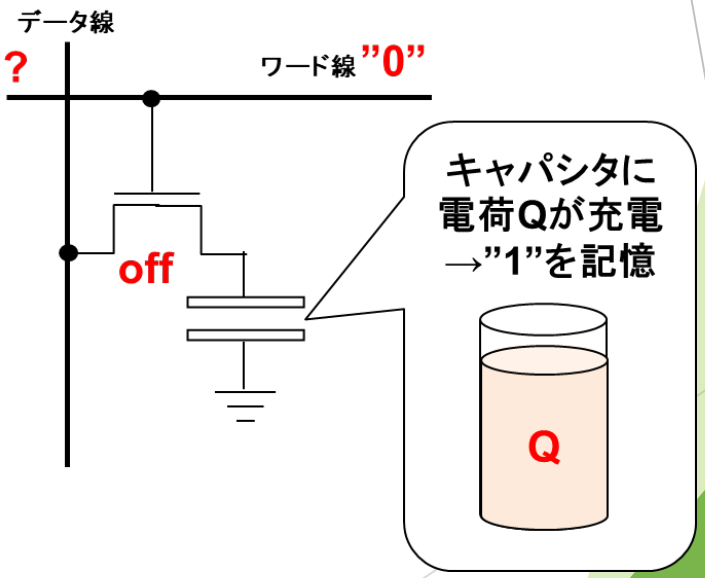
DRAMセルの記憶メカニズム

- データ”1”読み出し

⑥ ワード線を”1”から”0”へ（トランジスタをonからoffへ）



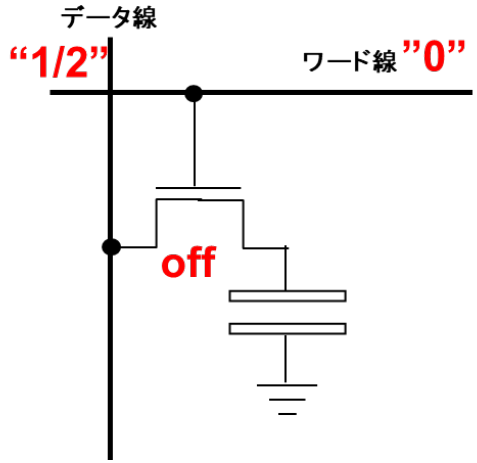
⑦ データ線からデータ”1”を消去。記憶を保持



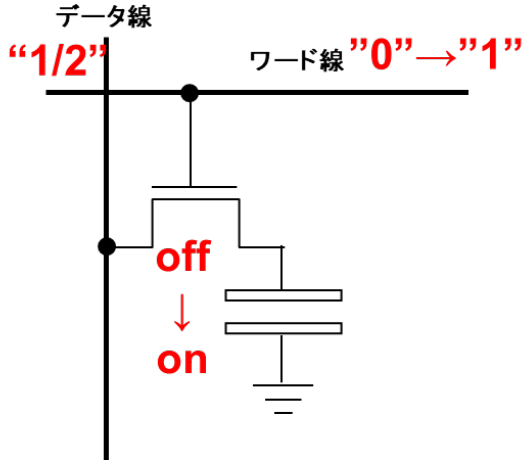
DRAMセルの記憶メカニズム

- データ”0”読み出し

① データ線にデータ”1/2”（”0”と”1”の中間値）を印加



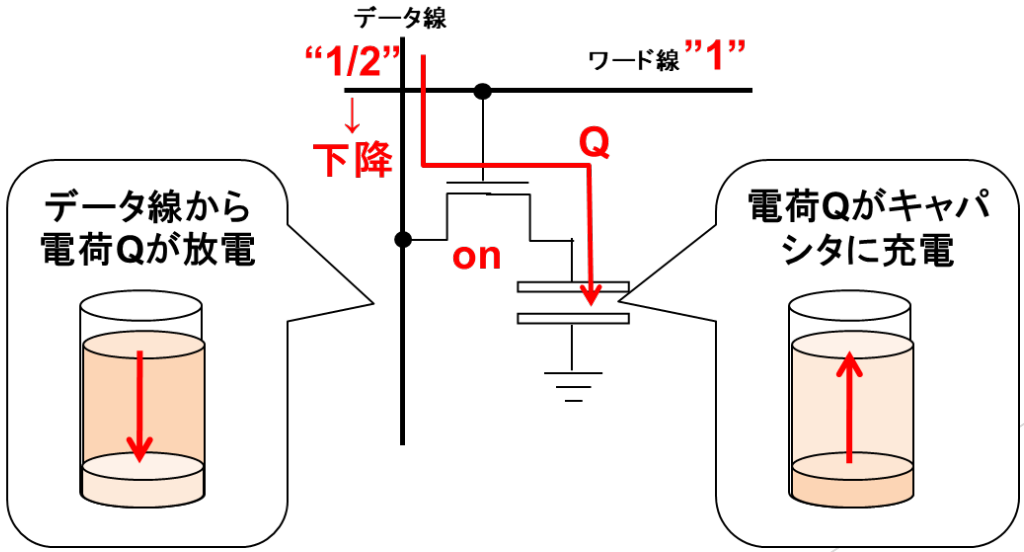
② ワード線を”0”から”1”へ（トランジスタをoffからonへ）



DRAMセルの記憶メカニズム

- データ”0”読み出し

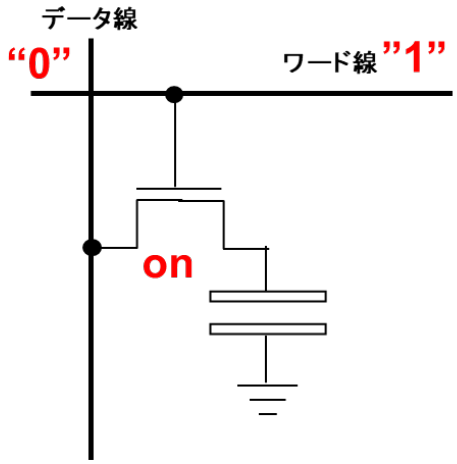
③ データ線からキャパシタに電荷Qが流入
→データ線の電圧が下降
→データ”0”を記憶していたと判断



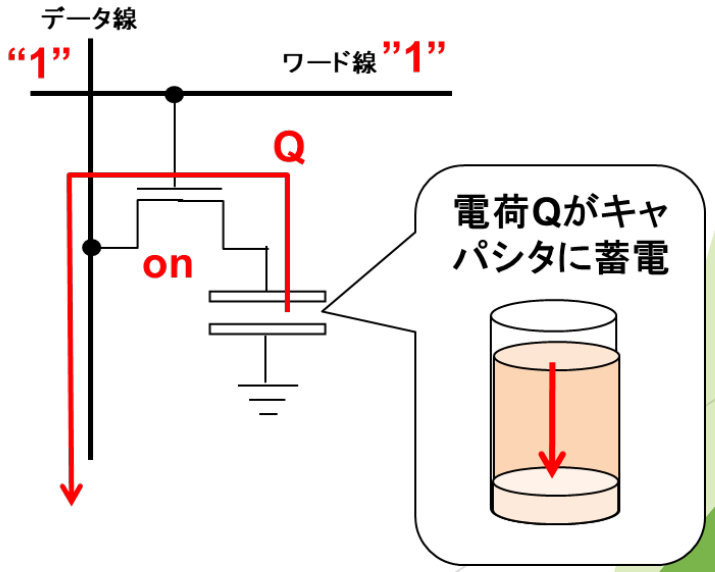
DRAMセルの記憶メカニズム

- データ”0”読み出し

④ 再びデータ”0”を書き込むために、データ線にデータ”0”を印加



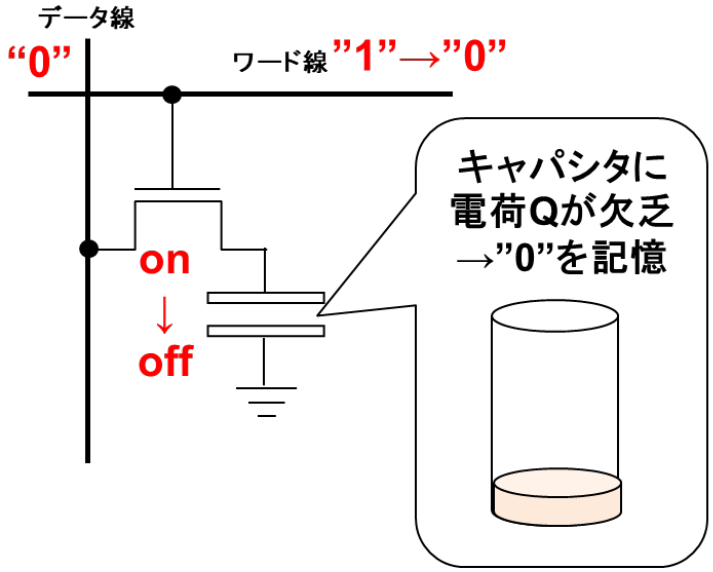
⑤ キャパシタからデータ線に電荷Qが流出



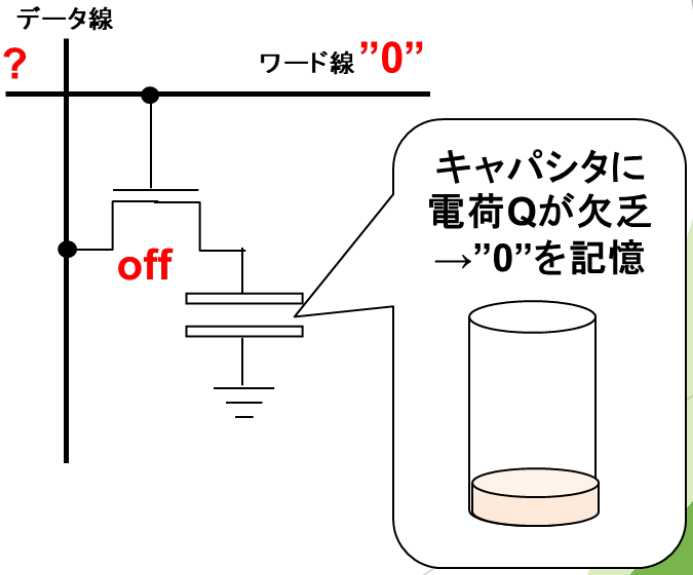
DRAMセルの記憶メカニズム

- データ”0”読み出し

⑥ ワード線を”1”から”0”へ（トランジスタをonからoffへ）



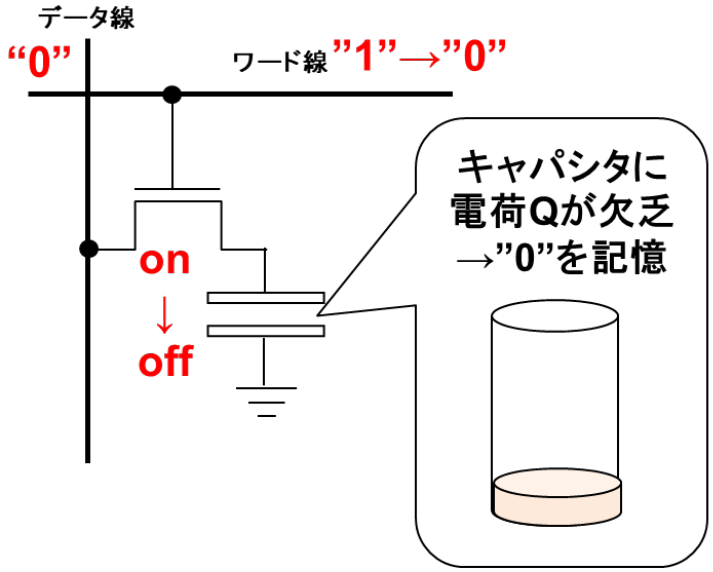
⑦ データ線からデータ”0”を消去。記憶を保持



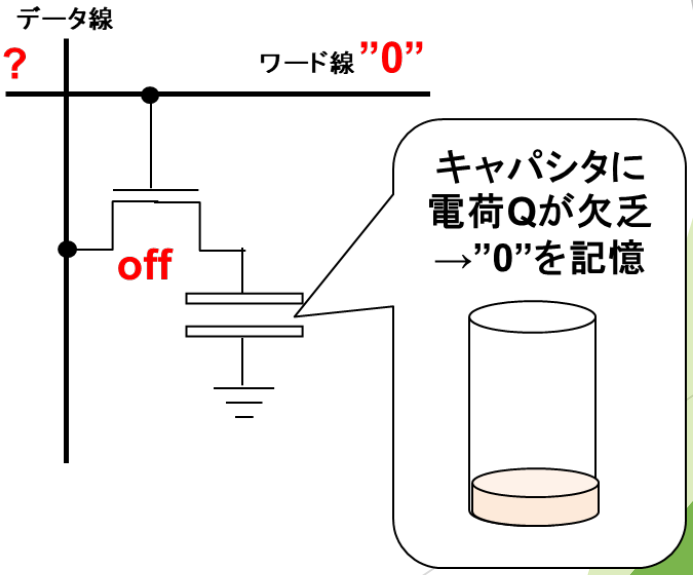
DRAMセルの記憶メカニズム

- データ”0”読み出し

⑥ ワード線を”1”から”0”へ（トランジスタをonからoffへ）

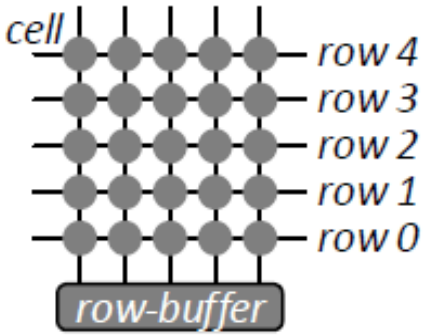


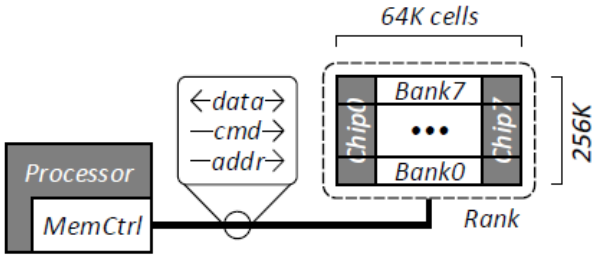
⑦ データ線からデータ”0”を消去。記憶を保持



DRAMのリフレッシュ

- DRAMセルは時間がたつと情報が失われる
  - キャパシタにチャージした電荷が抜けるため
  - 定期的な読出し、書戻しが必要（＝リフレッシュ）
- Row-buffer を使ったリフレッシュ
  - データを行ごとにリフレッシュする
    →DRAMへのアクセスは行単位で行う
- Retention time
  - DRAMセルがデータを保持できる時間
    DDR3 の仕様では 64 [ms]
  - この時間以内に前セルのリフレッシュが必要
  - リフレッシュもメモリコントローラの役割の一つ





Rowhammerについて

- 関連：CVE-2015-0565、CVE-2016-6728
- 概要
  - DRAMアクセスの副作用により、アクセス先以外の近くの行の内容が書き換わる可能性がある現象を利用
- 攻撃方法
  - 悪意あるプログラムの実行
    - Androidアプリ
    - Javascript
- 影響範囲
  - 2011年以降に製造されたDRAMモジュールを利用したシステム

Rowhammer現象の原理 1

- リフレッシュ間隔よりも高頻度にあるワードラインの電圧を繰り返し変化させると近くの行のセルに副作用がある
  - ワードライン同士の電磁気的な相互作用が原因であることが分かっている
  - 主な原因はプロセスの微細化
- 複数回の作用が累積し、0/1の閾値を超えると値が変化する

Rowhammer現象の原理 2

- どちらが攻撃プログラムか？

キャッシュライン
フラッシュ命令

```
1 code1a:
2   mov (X), %eax
3   mov (Y), %ebx
4   clflush (X)
5   clflush (Y)
6   mfence
7   jmp code1a
```

```
1 code1b:
2   mov (X), %eax
3   clflush (X)
4
5
6   mfence
7   jmp code1b
```

- ヒント：メモリアクセスにはRow-bufferを使用

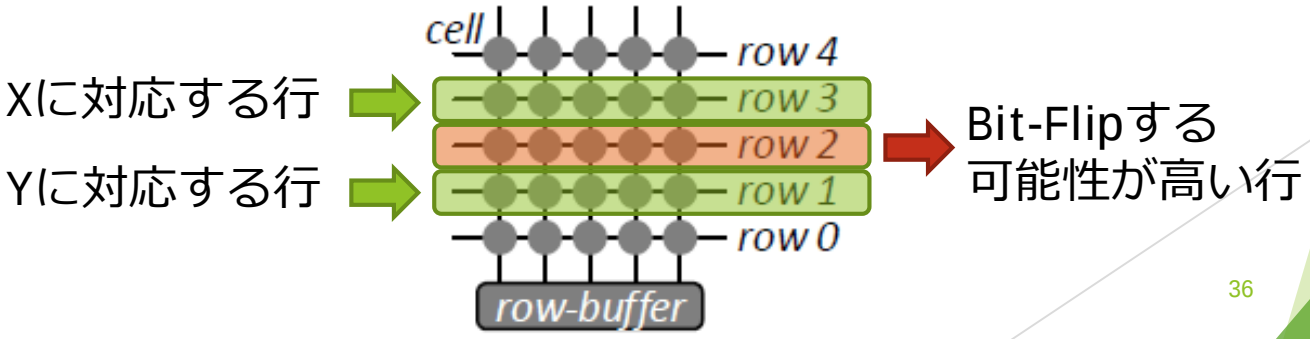
Rowhammer現象の原理 2

- どちらが攻撃プログラムか？

```
1 code1a:
2   mov (X), %eax
3   mov (Y), %ebx
4   clflush (X)
5   clflush (Y)
6   mfence
7   jmp code1a
```

```
1 code1b:
2   mov (X), %eax
3   clflush (X)
4
5
6   mfence
7   jmp code1b
```

  - ヒント：メモリアクセスにはRow-bufferを使用



Rowhammer現象の実証 1

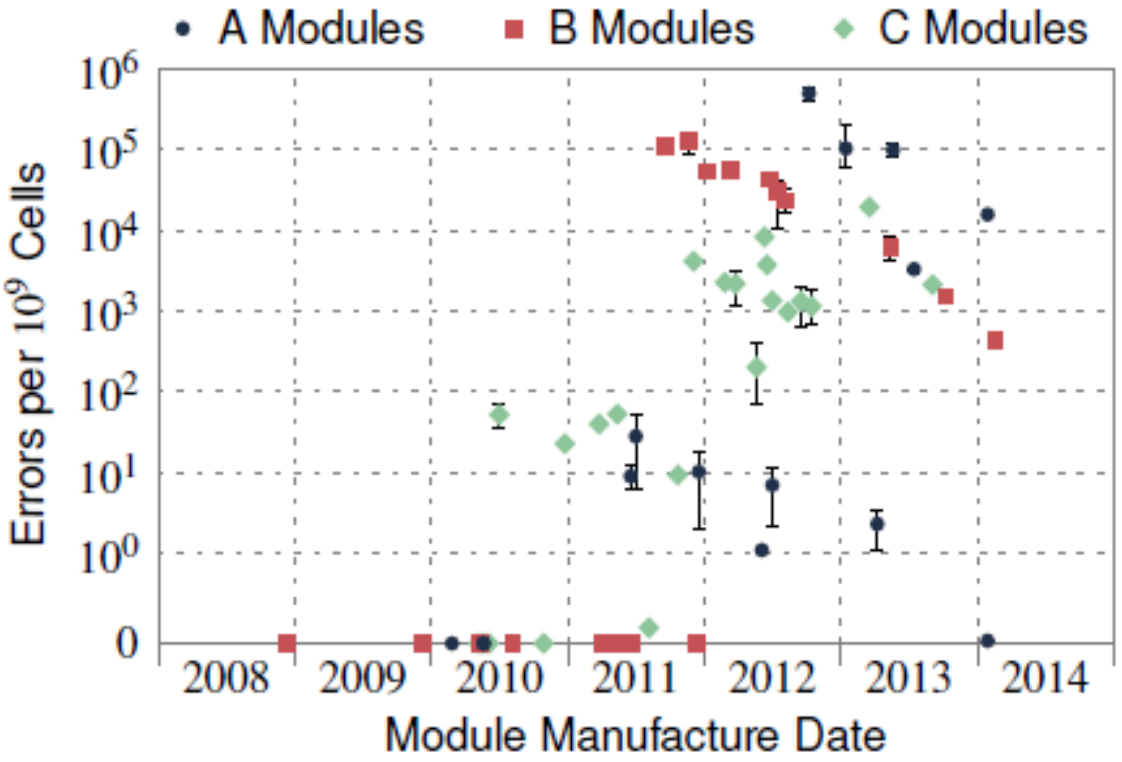
- Bit-Flipの回数

Bit-Flip	Sandy Bridge	Ivy Bridge	Haswell	Piledriver
'0' → '1'	7,992	10,273	11,404	47
'1' → '0'	8,125	10,449	11,467	12

Intel CPUの方が頻発
CPU世代が新しい方が起きやすい

Rowhammer現象の実証 2

- チップ製造メーカと年式とBit-Flip回数の関係



Rowhammer攻撃の例

- NaCl Sandbox エスケープ [‘15]
  - サンドボックスから抜け出し、システムコールを直接発行できる可能性が増大
  - NaClでclflush命令を実行できないように修正
- カーネル権限昇格 [‘15]
  - ページテーブルエントリにbit-flipを起こすことでコンピュータ上の物理メモリにアクセス可能に
  - clflushは特権命令ではないので防ぐのは困難
- Rowhammer.js
  - JavaScriptによる実装。特定の命令セットアーキテクチャに依存せずにキャッシュラインの破棄が起こるコードの実証
- Androidアプリ：DRAMMER [‘16]
  - Rowhammer現象を使って特権を取得するアプリ

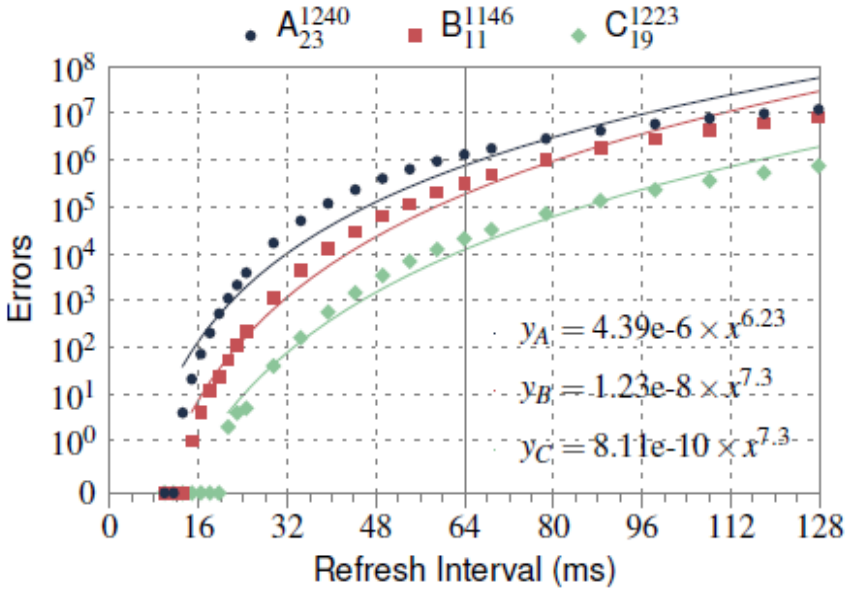
Rowhammer対策 1

- ECC (Error Correction Codes) メモリの利用
  - DRAMのECCはSECDED (Single Error-Correction Double Error-Detection)
  - 複数ビットがフリップするケースがあり、ECCで守れば安心とは言えない状況

Module	*Number of 64-bit words with X errors*			
	$X = 1$	$X = 2$	$X = 3$	$X = 4$
A_{23}	9,709,721	**181,856**	**2,248**	**18**
B_{11}	2,632,280	**13,638**	**47**	0
C_{19}	141,821	**42**	0	0

Rowhammer対策 2

- リフレッシュ間隔の変更
  - 仕様上64msのリフレッシュ間隔を短くすれば攻撃の可能性はかなり減少



  - リフレッシュ処理の量が増加するため性能に影響あり

Rowhammerまとめ

- マイクロアーキテクチャの実装によりアーキテクチャステートに副作用のある操作が可能
  - DRAMチップの微細化に伴うワードライン間の相互作用顕著化を利用
  - 行に高頻度にアクセスすると近くの行のデータが化ける
  - より最近のDRAMチップほど起きやすい傾向
- 攻撃への利用
  - サンドボックスエスケープ
  - 権限昇格
- 対策
  - ECC は有効だが完全ではない
  - リフレッシュ間隔を狭めることが有効

マイクロアーキテクチャ攻撃の影響と対応

マイクロアーキテクチャ攻撃の影響

- マイクロアーキテクチャレベルの振る舞いはコンピュータ・システム設計の根幹
  - ソフトウェアレイヤの安全性もハードウェアの動作に基づいて設計・評価されている
  - マイクロアーキテクチャレベルの脆弱性が発見されると影響範囲が広く、影響範囲のアセスメントコスト大
  - ハードウェア入れ替えは難しいため、根本的な解決ができない場合がある
  - 攻撃検出が難しい
- ソフトウェアから利用可能な脆弱性
  - ハードウェアへの接触なし
    - ブラウザ、クラウド、スマートフォンアプリ

マイクロアーキテクチャ攻撃への対応

- システムソフトウェア（OS、コンパイラ）での緩和策が有効な場合がある
  - カーネルアップデートや、プログラムの再コンパイルが必要
  - システム設計、運用時にソフトウェアスタックの更新を考慮しておく
- 性能とセキュリティのトレードオフ
  - 緩和策の中には性能低下を伴うものがある
  - 攻撃の影響を考慮して適用の是非を判断

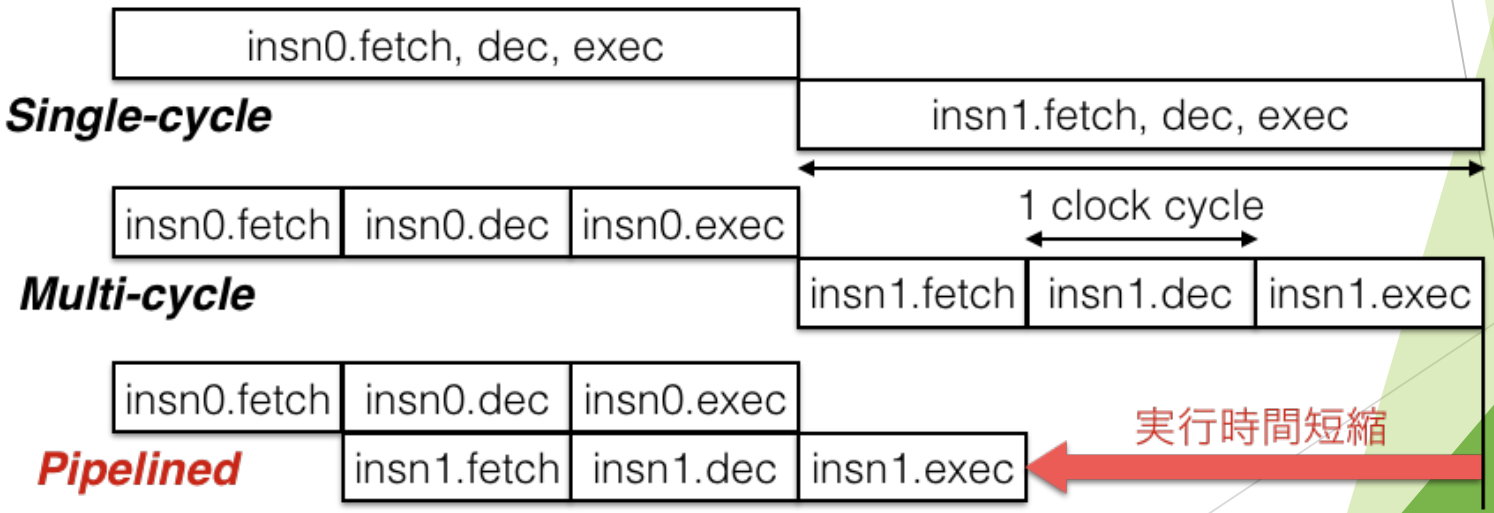
前回のおさらい（Spectre）

アウトオブオーダプロセッサ

- プロセッサ高性能化技術の結晶
  - パイプライン化
  - スーパスカラ
  - アウトオブオーダ命令実行

プロセッサのパイプライン化

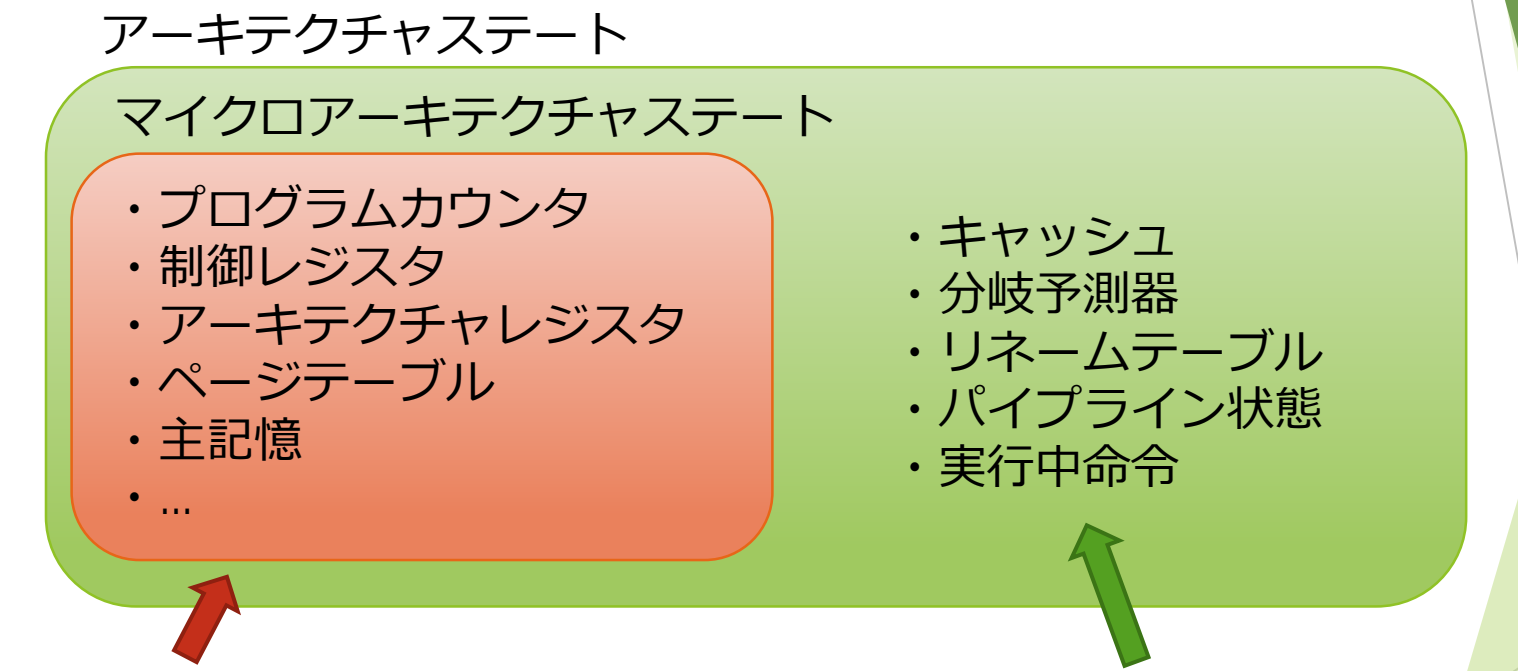
- プロセッサのパイプライン化
  - プロセッサの基本的な高速化手法
  - プロセッサをN段のパイプラインで構成
    →（理想的には）周波数N倍→スループットN倍
- パイプライン化の例



分岐予測による投機実行

- 分岐命令の実行よりも早いステージで分岐するか否かを予測し、後続命令のフェッチを開始する手法
- 予測が外れた場合はプロセッサの状態を戻して正しい命令を実行し直す
- 分岐予測が当たればパイプラインを埋めることができて性能向上可能

アーキテクチャステートとマイクロアーキテクチャステートの関係

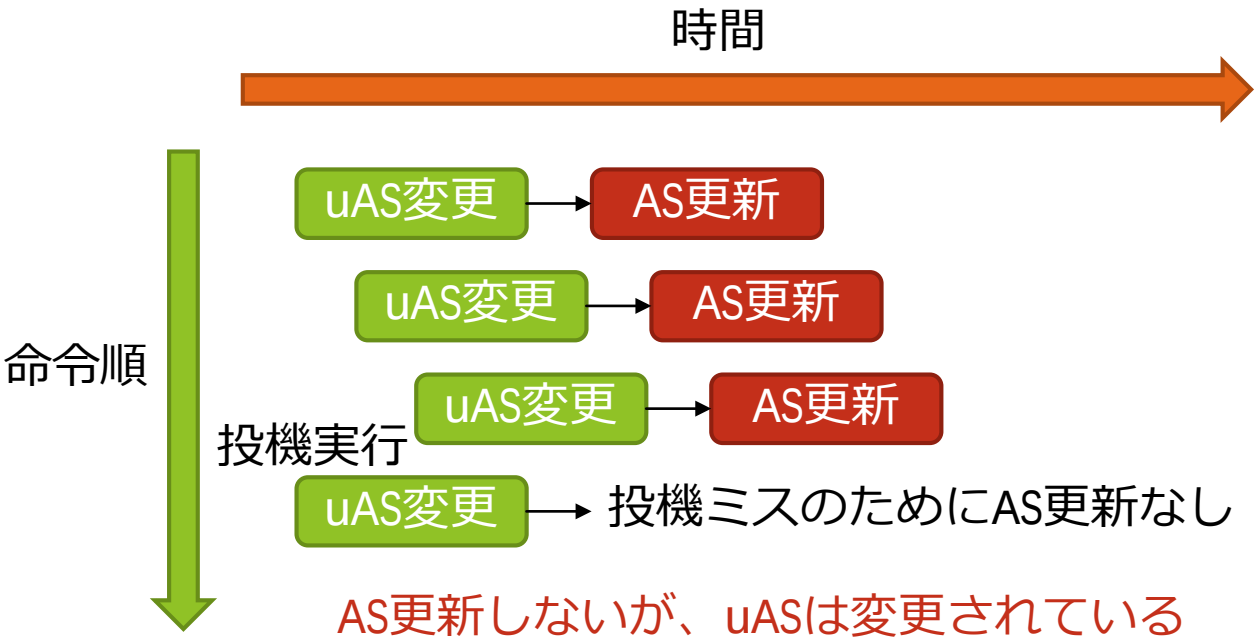


サイドチャネル攻撃

- 装置の動作状況を様々な物理的手段で観察することにより、装置内部のセンシティブな情報を取得しようとする攻撃方法の総称

- 分類
  - タイミング攻撃
    - 処理時間の違いに着目
  - 電力解析攻撃
    - 消費電力の違いに着目
  - 電磁波解析攻撃
    - 装置から漏洩する電磁波に着目

投機実行を使った攻撃



AS更新しないが、uASは変更されている

- 本来アクセスできない攻撃対象の情報をuASに残す
- uASの変更をサイドチャネル攻撃で検出

Spectre攻撃の原理１：概要

1. あるプロセスの投機的実行で、読まれてはいけない情報をCPUのキャッシュメモリに読み込むように仕向ける
2. 別プロセスでキャッシュの情報を参照することで、読まれてはいけない情報を読むことが可能となる

Spectre攻撃の原理２

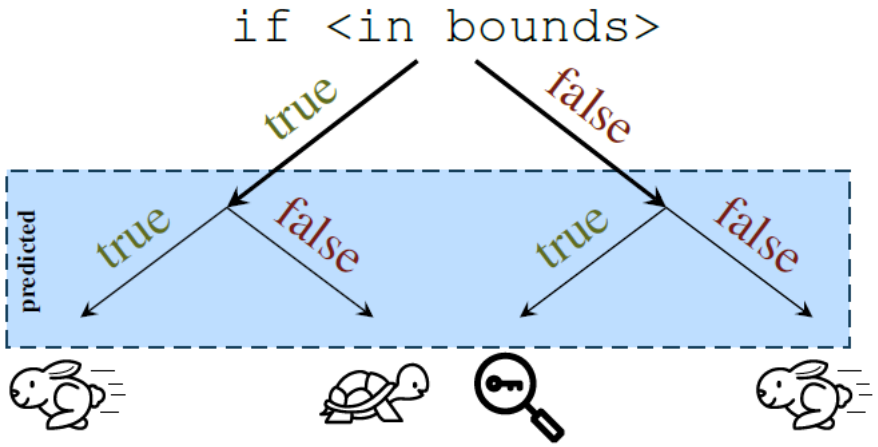
- 攻撃１：分岐予測ミスを利用した攻撃
  - 以下のコードを考えてみましょう

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

    - array1の各要素は１Ｂを想定
  - まずは分岐がTakenに予測されるよう分岐予測器を学習させた状態を想定
  - その後、xの値を書き換えるとどうなるか？
  - 投機的にアクセスするアドレス値
    - array2_base + [array1_base + x] × 4096
  - どのアドレスのデータがキャッシュに載っているか調べることでxの値が分かる

Spectre攻撃の原理 3

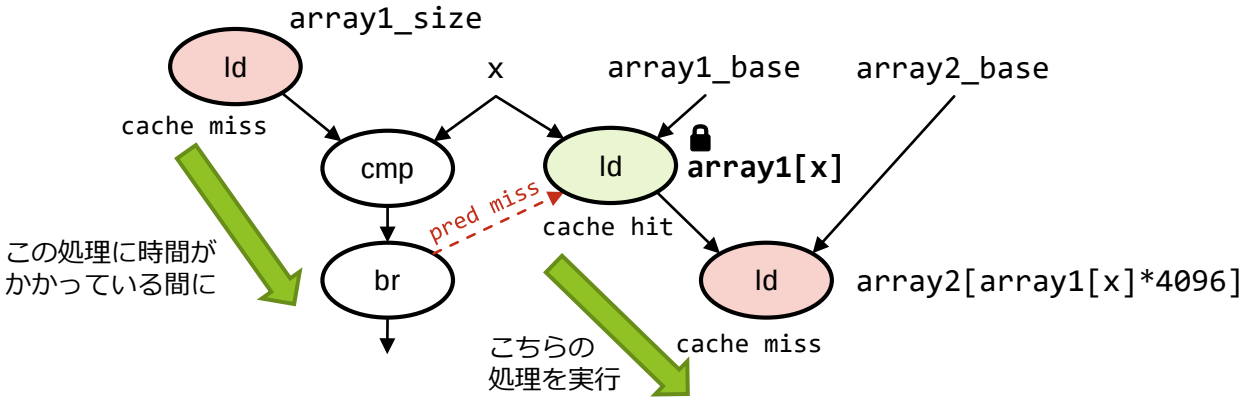
- 分岐予測と攻撃の関係



  - 分岐予測が当たった時に速くなるだけならよかったが、予測ミス時に特定の場合にキャッシュにデータ値に対応した痕跡が残る

Spectre攻撃の原理４

- 攻撃１成立の条件
  - xの値をarray1[x]が対象プログラムの秘密情報アドレスに対応するように設定可能なこと
  - array1_sizeとarray2がキャッシュされておらず、秘密情報がキャッシュされていること



  - xに関する条件分岐がtakenに予測なるよう分岐予測器を学習させていること

Spectre攻撃の原理 5

- 攻撃 1 のポイント
  - xの値さえ変更できれば、残りのデータアクセスするプログラム自体は既存のものを利用できる
  - 利用できる条件分岐の例
    - アクセス範囲チェック（既出）
    - 前にアクセスしたデータのチェック
    - 前にアクセスしたオブジェクトの型チェック
  - 利用できる投機実行処理の例
    - チェック結果をメモリに書く処理
    - 複雑な（たくさんの）命令を実行する処理