生命の科学で未来をつなぐ

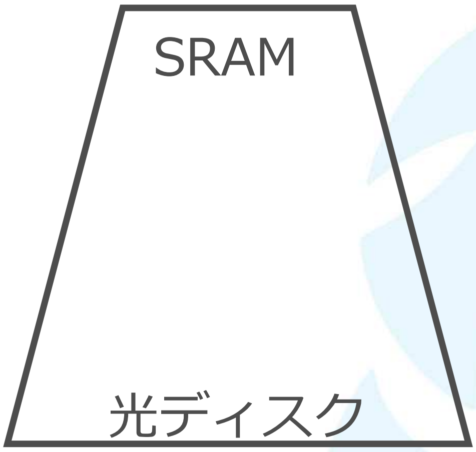
メモリ階層とキャッシュ

メモリ階層の考え方

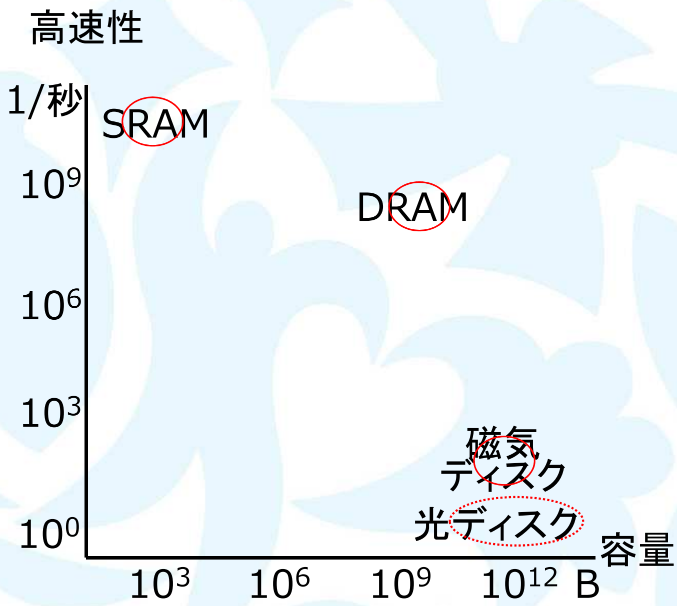
メモリ階層

メモリ素子

高速だが小容量



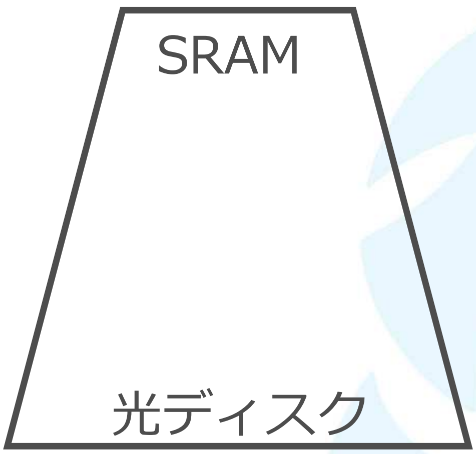
低速だが大容量



メモリ階層

メモリ素子

高速だが小容量



低速だが大容量

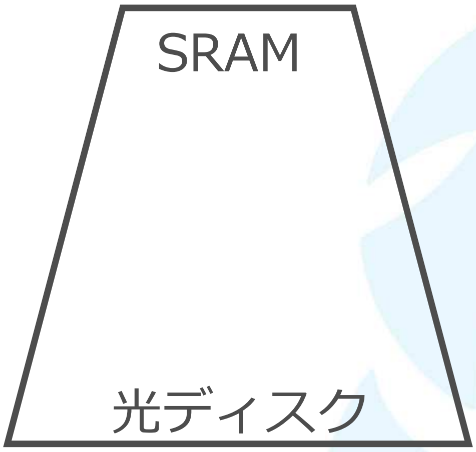
⇨

適材適所で使う

組合せて使う

メモリ階層

高速だが小容量



低速だが大容量

適材適所

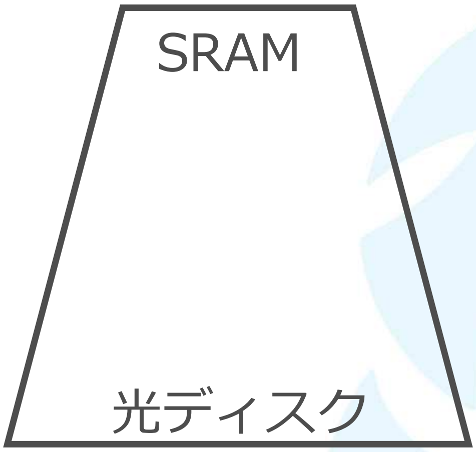
普段使い用
- CPUから頻繁にアクセス
- CPU速度で返答
- 少量で我慢

保存用
- 大量に保存
- 滅多に見ない

東邦大学

メモリ階層

高速だが小容量



低速だが大容量

適材適所

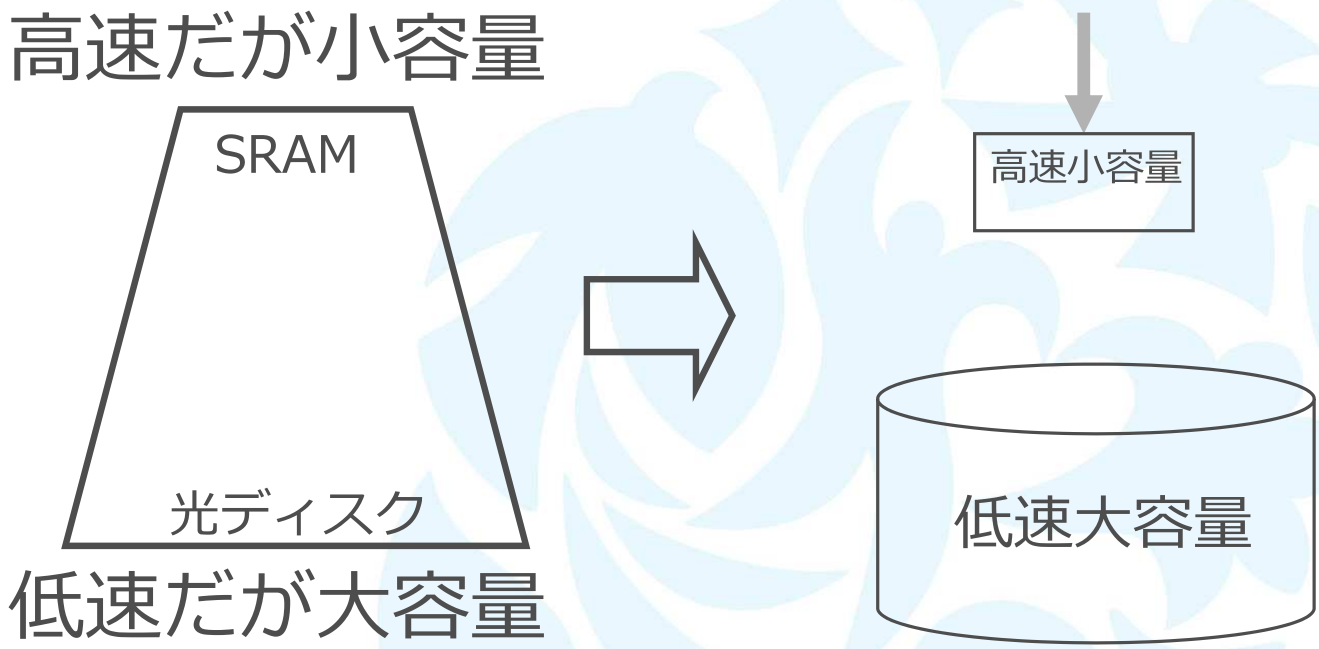
普段使い用
- レジスタ
- キャッシュ(後述)
- 主記憶
  - ⇒命令毎アクセス

保存用
- ファイル
  - ⇒保存用

東邦大学

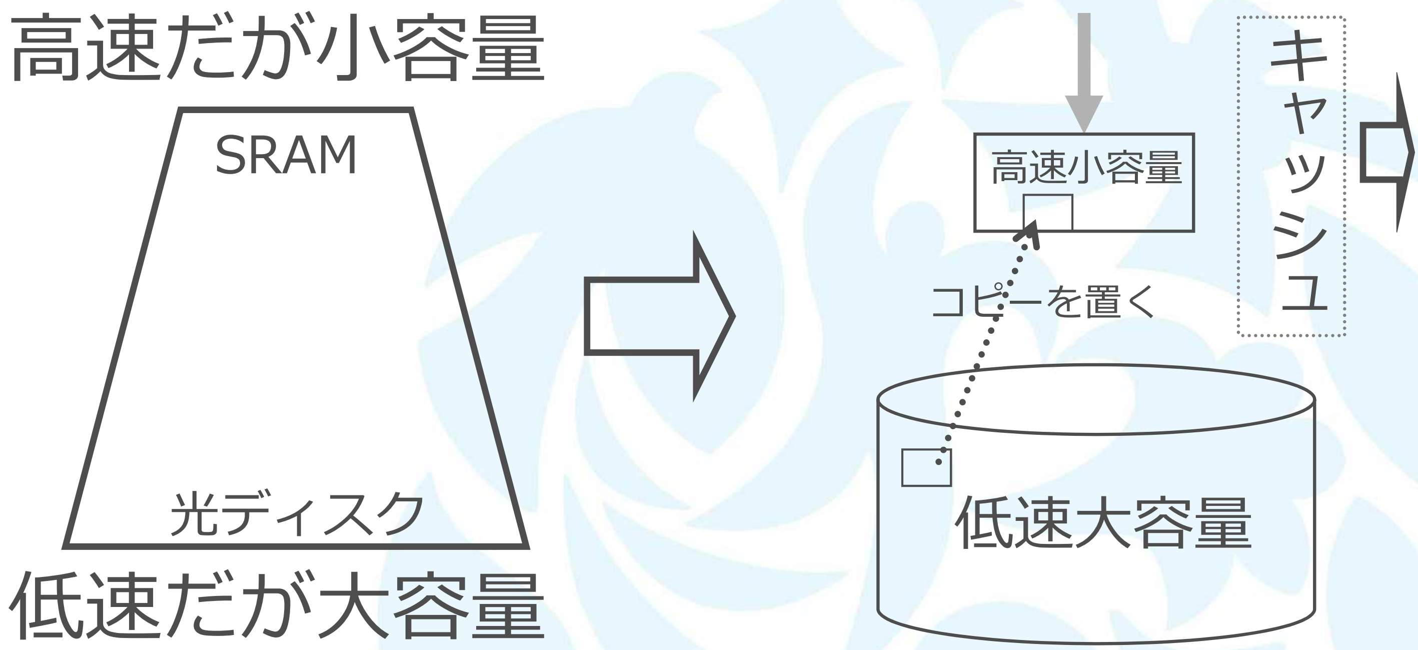
メモリ階層

組合せて使う

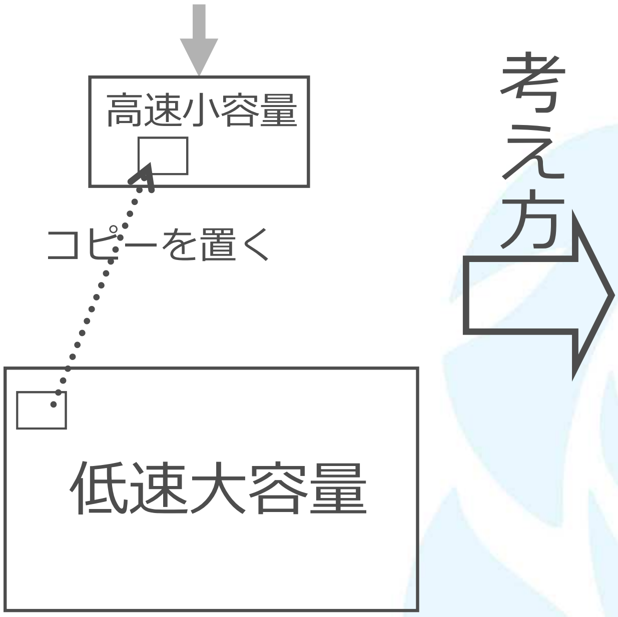


メモリ階層

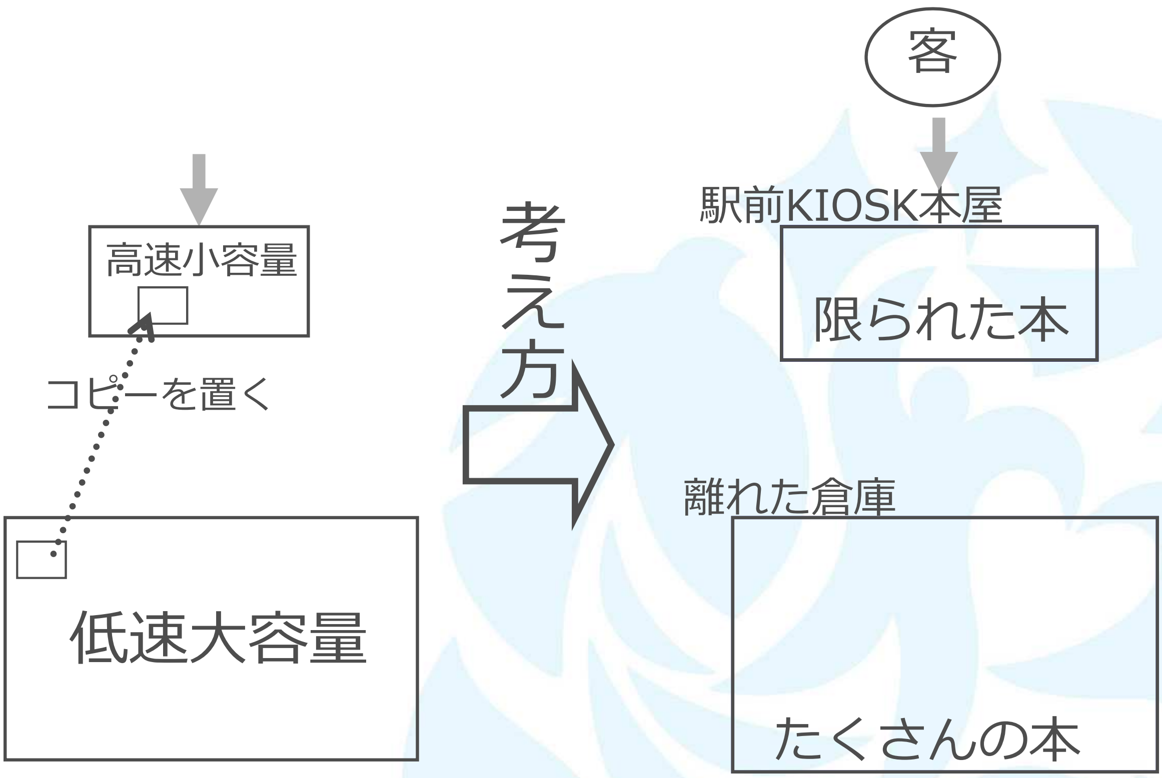
組合せて使う



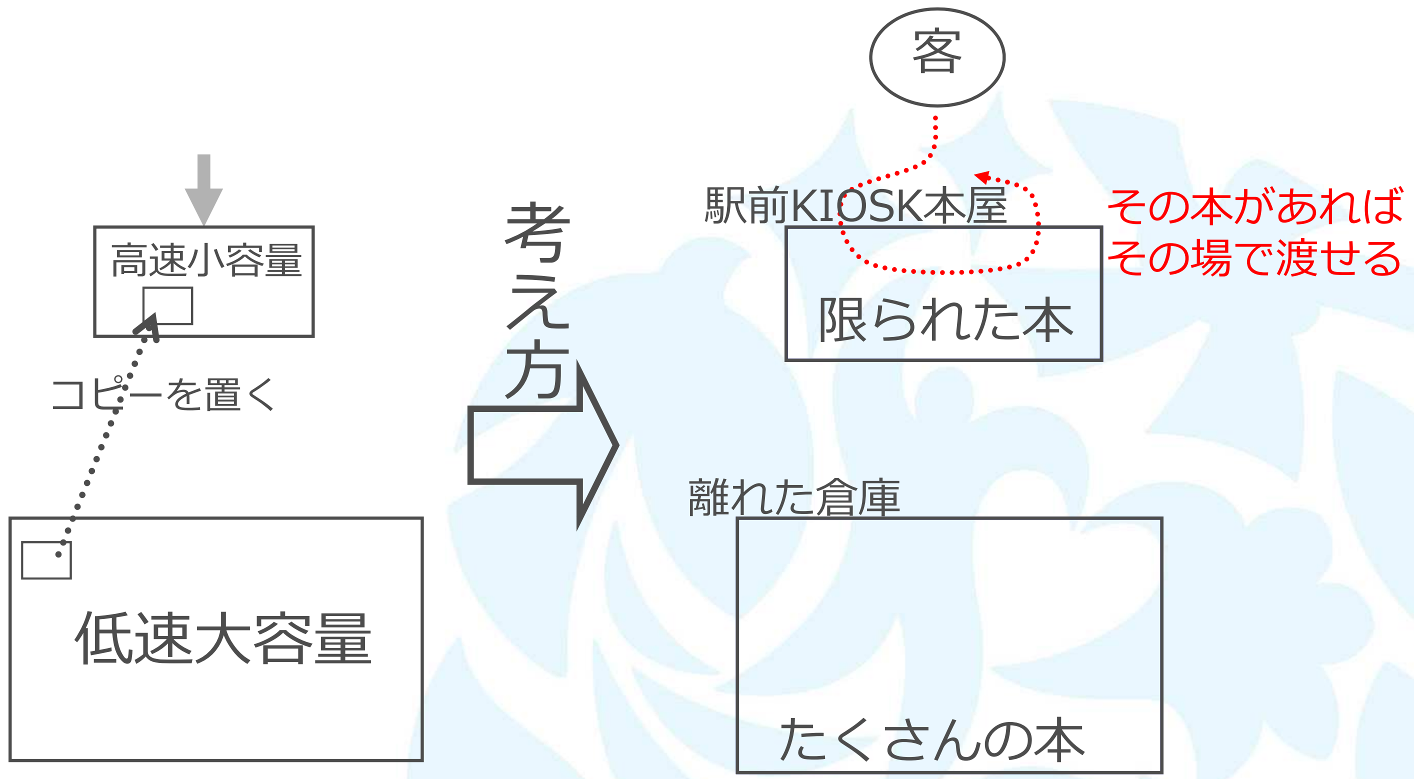
キャッシュ


高速小容量
コピーを置く
低速大容量
考え方

キャッシュ

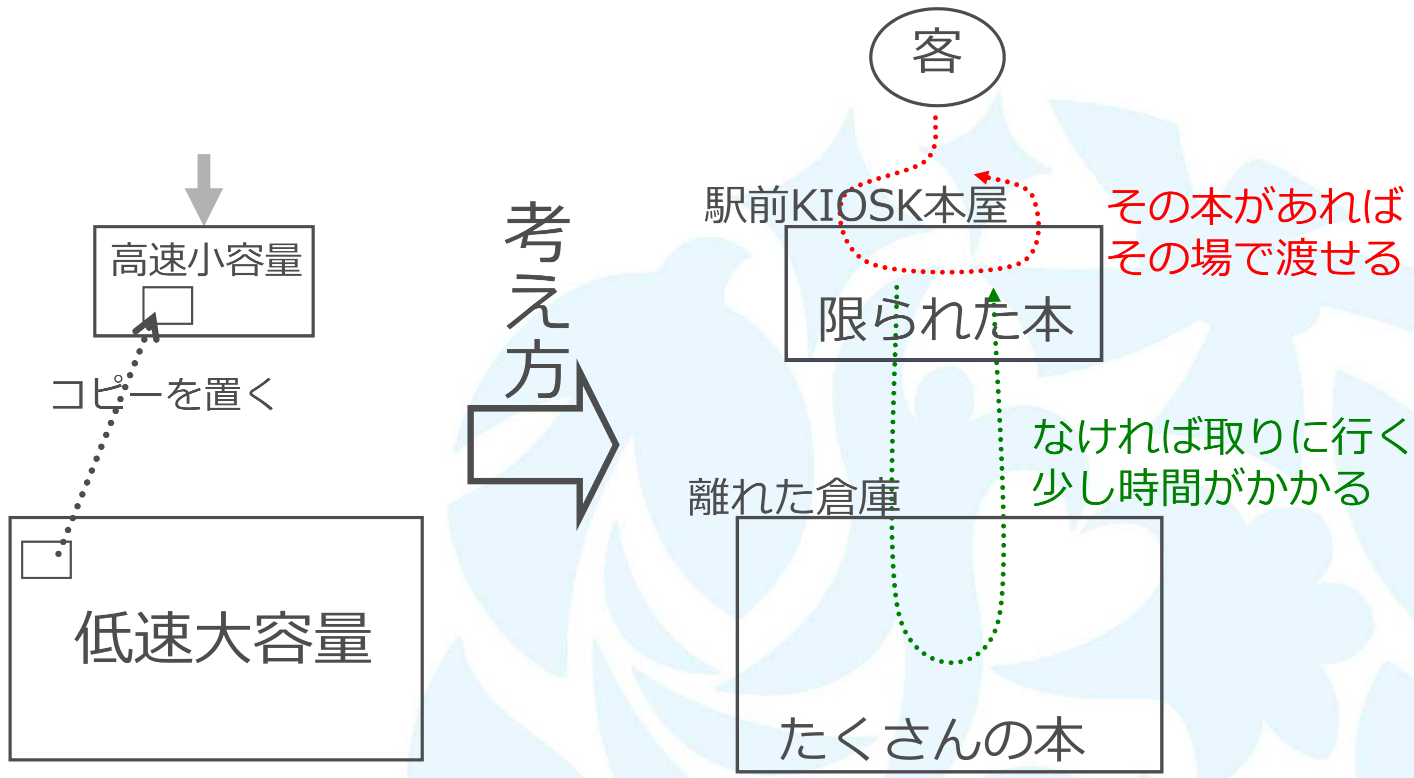

高速小容量
コピーを置く
低速大容量
考え方
客
駅前KIOSK本屋
限られた本
離れた倉庫
たくさんの本

キャッシュ



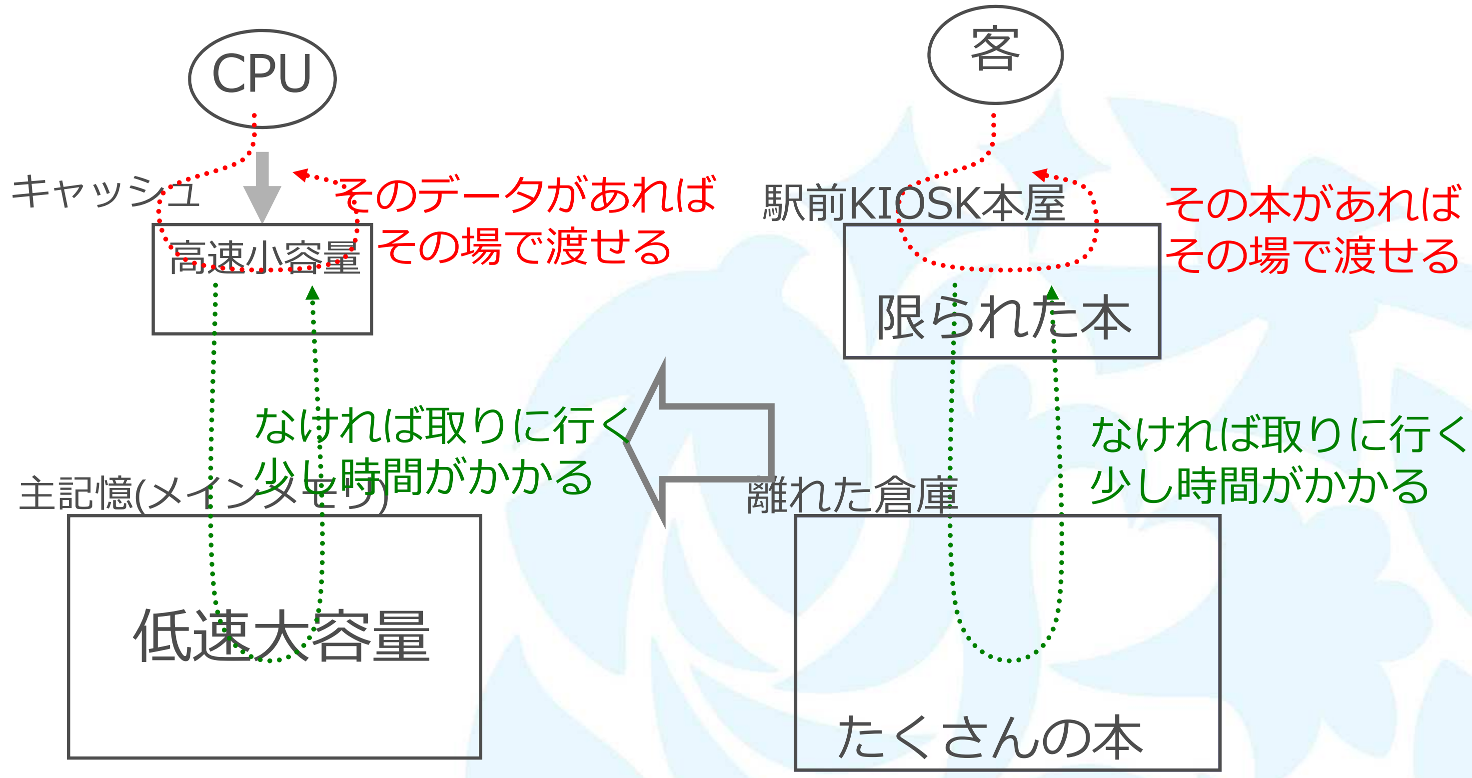
東邦大学

キャッシュ



東邦大学

キャッシュ



整理すると

CPU ⇒ キャッシュ ⇒ 主記憶

キャッシュには、主記憶の一部のコピー

CPUがキャッシュへ取りに来て
　あったらその場で返す　⇒　速い
　なかったら主記憶から持ってきて渡す
　　⇒　時間がかかる

キャッシュの仕組みのテストです

整理すると

CPU ⇒ キャッシュ ⇒ 主記憶

キャッシュには、主記憶の一部のコピー

CPUがキャッシュへ取りに来て
あったらその場で返す　⇒　速い
なかったら主記憶から持ってきて渡す
⇒　時間がかかる

キャッシュは

　　　と　　　の間にあり

　　　よりはずっと容量が

主記憶の一部を覚えており

　　がデータを要求したとき

　そのデータを持っていれば

　無ければ

キャッシュは

　CPUと主記憶の間にあり

　主記憶よりはずっと容量が小さい

主記憶の一部を覚えており

　CPUがデータを要求したとき

　そのデータを持っていれば返し

　無ければ主記憶に取りにいく

キャッシュの動作がわかりましたか？

○ ×

次へ

なぜこれがうまくいくのか？

なぜこれがうまくいくのか？

ほとんどの客がKIOSKで済めばいい

なぜこれがうまくいくのか？

ほとんどの客がKIOSKで済めばいい

ほとんどの客が買う本が限られて(売れ筋)いて
その本がKIOSKに置いてあればよい

もう少しきちんと考えてみよう

アクセス時間のモデル

KIOSKにあったらその場で返す
　なかったら倉庫から持ってきて渡す

二者択一　どちらか一方が起こる

アクセス時間のモデル

KIOSKにあったらその場で返す
　なかったら倉庫から持ってきて渡す

二者択一　どちらか一方が起こる

かかる時間の期待値（予測値）は

東邦大学

アクセス時間のモデル

KIOSKにあったらその場で返す
　なかったら倉庫から持ってきて渡す

二者択一　どちらか一方が起こる

かかる時間の期待値（予測値）は

(KIOSKで済む場合の時間)×(済む確率)

東邦大学

アクセス時間のモデル

KIOSKにあったらその場で返す
なかったら倉庫から持ってきて渡す

二者択一　どちらか一方が起こる

かかる時間の期待値（予測値）は

(KIOSKで済む場合の時間)×(済む確率)
＋ (済まない場合の時間)×(済まない確率)

東邦大学

アクセス時間のモデル

かかる時間の期待値（予測値）は

(KIOSKで済む場合の時間)×(済む確率)
＋ (済まない場合の時間)×(済まない確率)

東邦大学

アクセス時間のモデル

かかる時間の期待値（予測値）は

(KIOSKで済む場合の時間)×(済む確率)
＋ (済まない場合の時間)×(済まない確率)

もし

	かかる時間	確率
KIOSKで済む場合	1	0.5
済まない場合	10	0.5

⇨

アクセス時間のモデル

かかる時間の期待値（予測値）は

(KIOSKで済む場合の時間)×(済む確率)
＋ (済まない場合の時間)×(済まない確率)

もし

	かかる時間	確率
KIOSKで済む場合	1	0.5
済まない場合	10	0.5

⇨

期待値
$= 1 \times 0.5$
$+ 10 \times 0.5$
$= 5.5$

アクセス時間のモデル

かかる時間の期待値（予測値）は

(KIOSKで済む場合の時間)×(済む確率)
＋ (済まない場合の時間)×(済まない確率)

もし

	かかる時間	確率
KIOSKで済む場合	1	0.5
済まない場合	10	0.5

期待値
＝ 1 × 0.5
＋10 × 0.5
＝ 5.5　速くない

アクセス時間のモデル

かかる時間の期待値（予測値）は

(KIOSKで済む場合の時間)×(済む確率)
＋ (済まない場合の時間)×(済まない確率)

もし

	かかる時間	確率
KIOSKで済む場合	1	0.9
済まない場合	10	0.1

ほとんど
KIOSKで済む
とどうなるか

アクセス時間のモデル

かかる時間の期待値（予測値）は

(KIOSKで済む場合の時間)×(済む確率)

＋ (済まない場合の時間)×(済まない確率)

もし

	かかる時間	確率
KIOSKで済む場合	1	0.9
済まない場合	10	0.1

期待値

= 1 × 0.9

＋10 × 0.1

= 1.9　2倍ぐらい

アクセス時間のモデル

かかる時間の期待値（予測値）は

(KIOSKで済む場合の時間)×(済む確率)

＋ (済まない場合の時間)×(済まない確率)

もし

	かかる時間	確率
KIOSKで済む場合	1	0.99
済まない場合	10	0.01

もっとほとんどKIOSKで済むとどうなるか

アクセス時間のモデル

かかる時間の期待値（予測値）は

(KIOSKで済む場合の時間)×(済む確率)
+ (済まない場合の時間)×(済まない確率)

もし

	かかる時間	確率
KIOSKで済む場合	1	0.99
済まない場合	10	0.01

期待値
= 1 × 0.99
+ 10 × 0.01
= 1.09 1.1倍弱

アクセス時間のモデル

まとめると、時間比率が 1 対 10 のとき

KIOSKで済む場合の確率	かかる時間 (KIOSKだけの時間との比)
0.5	5.5
0.9	1.9
0.99	1.09

アクセス時間のモデル

まとめると、時間比率が 1 対 10 のとき

KIOSKで済む場合の確率	かかる時間(KIOSKだけの時間との比)
0.5	5.5
0.9	1.9
0.99	1.09

⇒

KIOSKだけで済む場合の確率(ヒット率)が十分に大きければ
(ここでは0.99)
倉庫に取り行くことがあっても、あまり時間は増えない (1.09)

東邦大学

アクセス時間のまとめ

キャッシュシステム全体のアクセス時間は

×

＋ ×

キャッシュにデータがある確率(ヒット率)が　　　とき初めて、
全体の時間が

東邦大学

アクセス時間のまとめ

キャッシュシステム全体のアクセス時間は

	(キャッシュアクセス時間)	×	(キャッシュにデータがある確率)
+	(主記憶のアクセス時間)	×	(ないので主記憶に取りに行く確率)

キャッシュにデータがある確率(ヒット率)が　　　　とき初めて、
全体の時間が

東邦大学

アクセス時間のまとめ

キャッシュシステム全体のアクセス時間は

	(キャッシュアクセス時間)	×	(キャッシュにデータがある確率)
+	(主記憶のアクセス時間)	×	(ないので主記憶に取りに行く確率)

キャッシュにデータがある確率(ヒット率)が十分高いとき初めて、
全体の時間がキャッシュに近づく

東邦大学

キャッシュシステムの
アクセス時間のモデル
理解できましたか

次へ

では、ヒット率はどのぐらい？

ヒット率＝CPUからのアクセスが
(手前の)キャッシュで済む確率

ヒット率（手前だけで済む確率）

- 決まった値があるわけではない
  - メモリのアクセスの仕方に依存する

東邦大学

ヒット率（手前だけで済む確率）

- 決まった値があるわけではない
  - メモリのアクセスの仕方に依存する
    - 同じ場所に繰り返してアクセスすれば
      - 最初の1回は主記憶（倉庫）に取りに行くが
        - 次からはキャッシュ（KIOSK）で済む

東邦大学

ヒット率（手前だけで済む確率）

決まった値があるわけではない

メモリのアクセスの仕方に依存する

同じ場所に繰り返してアクセスすれば

最初の1回は主記憶（倉庫）に取りに行くが

次からはキャッシュ（KIOSK）で済む

たとえば100回繰り返してアクセスすれば

最初の1回は主記憶（倉庫）に取りに行くが

99回はキャッシュ（KIOSK）で済む

⇒　ヒット率は99/100＝0.99

東邦大学

ヒット率（キャッシュで済む確率）

同じ場所に繰り返してアクセスすれば

最初の1回は主記憶（倉庫）に取りに行くが

次からはキャッシュ（KIOSK）で済む

実は、持ってくる単位は1データ分ではなくて

大きいブロック（たとえば16データ分）なので

⇒　連続したアドレスにアクセスするなら

主記憶（倉庫）には16回中1回のみ

東邦大学

ヒット率（キャッシュで済む確率）

同じ場所に繰り返してアクセスすれば
最初の1回は主記憶（倉庫）に取りに行くが
次からはキャッシュ（KIOSK）で済む

実は、持ってくる単位は1データ分ではなくて
大きいブロック（たとえば16データ分）なので
⇒　連続したアドレスにアクセスするなら
主記憶（倉庫）には16回中1回のみ

連続した命令読出しは 連続したアドレス	ループを何度も回れば 命令もデータも同じアドレス

ヒット率（キャッシュで済む確率）

普通考えるより、同じようなアドレスを
繰り返しアクセスする確率が高い

ヒット率（キャッシュで済む確率）

普通考えるより、同じようなアドレスを
繰り返しアクセスする確率が高い

⇒　「アクセスの局所性」

東邦大学

ヒット率（キャッシュで済む確率）

普通考えるより、同じようなアドレスを
繰り返しアクセスする確率が高い

⇒　「アクセスの局所性」

実際のプログラムで測定してみると
かなり局所性が高いことが分かっている

東邦大学

ヒット率（キャッシュで済む確率）

普通考えるより、同じようなアドレスを
　繰り返しアクセスする確率が高い

⇒　「アクセスの局所性」

実際のプログラムで測定してみると
　かなり局所性が高いことが分かっている

⇒　キャッシュによって十分速くなる

東邦大学

ヒット率の性質のまとめ

ヒット率は
　普通考える(ランダム)より

理由は「アクセスの　　　　」が
　成り立つことが多いからである

キャッシュはブロック単位で転送され
　次の命令読出しやデータアクセスが
　同じブロック内に入る確率が

東邦大学

ヒット率の性質のまとめ

ヒット率は
　普通考える(ランダム)より高い
理由は「アクセスの　　　」が
　成り立つことが多いからである
キャッシュはブロック単位で転送され
　次の命令読出しやデータアクセスが
　同じブロック内に入る確率が

東邦大学

ヒット率の性質のまとめ

ヒット率は
　普通考える(ランダム)より高い
理由は「アクセスの局所性」が
　成り立つことが多いからである
キャッシュはブロック単位で転送され
　次の命令読出しやデータアクセスが
　同じブロック内に入る確率が

東邦大学

ヒット率の性質のまとめ

ヒット率は
普通考える(ランダム)より高い
理由は「アクセスの局所性」が
成り立つことが多いからである
キャッシュはブロック単位で転送され
次の命令読出しやデータアクセスが
同じブロック内に入る確率が高い

東邦大学

ヒット率の性質について
理解できましたか？

次へ

東邦大学