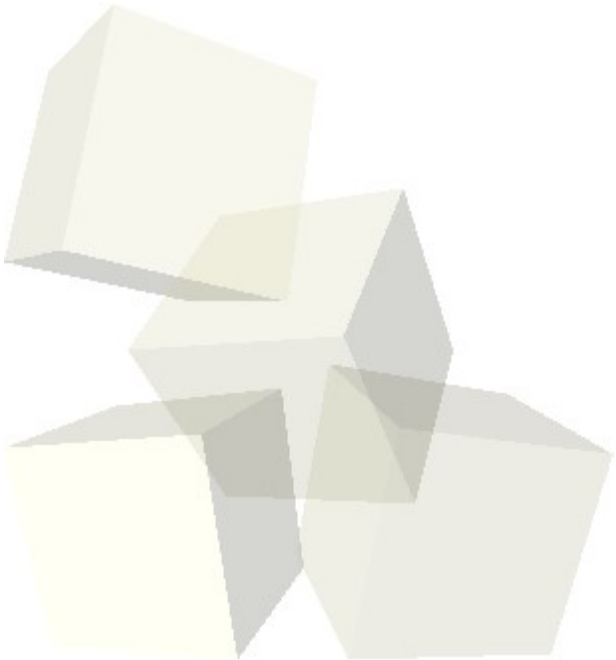
オペレーティングシステム

#3 CPUの仮想化：スケジューリング



復習：スレッド

- プロセス切り替え
  - コスト高
- スレッド
  - プロセスを、さらに細かい単位に分割
  - CPUリソースを割り当てる，さらに細かい単位
  - 主記憶領域が同じのため，切り替えコスト低

復習：割込み

- 割込み
  - 通常のCPU演算動作とは異なる事象
    - → キーボード入力を受け取った
    - → 自動車がどこかに衝突した
    - → サーバからデータが送られてきた
  - 割込み発生時にプロセスの切り替えが起こる
  - TSSでは、プロセス切り替えのために
    インターバルタイマーが定期的に割込みを発生

復習：割込みの種類

- 内部割込み
  - スーパバイザコール割込み
  - プログラムチェック（例外）割込み
- 外部割込み
  - 入出力割込み
  - タイマ割込み
  - マシンチェック割込み
  - リスタート割込み

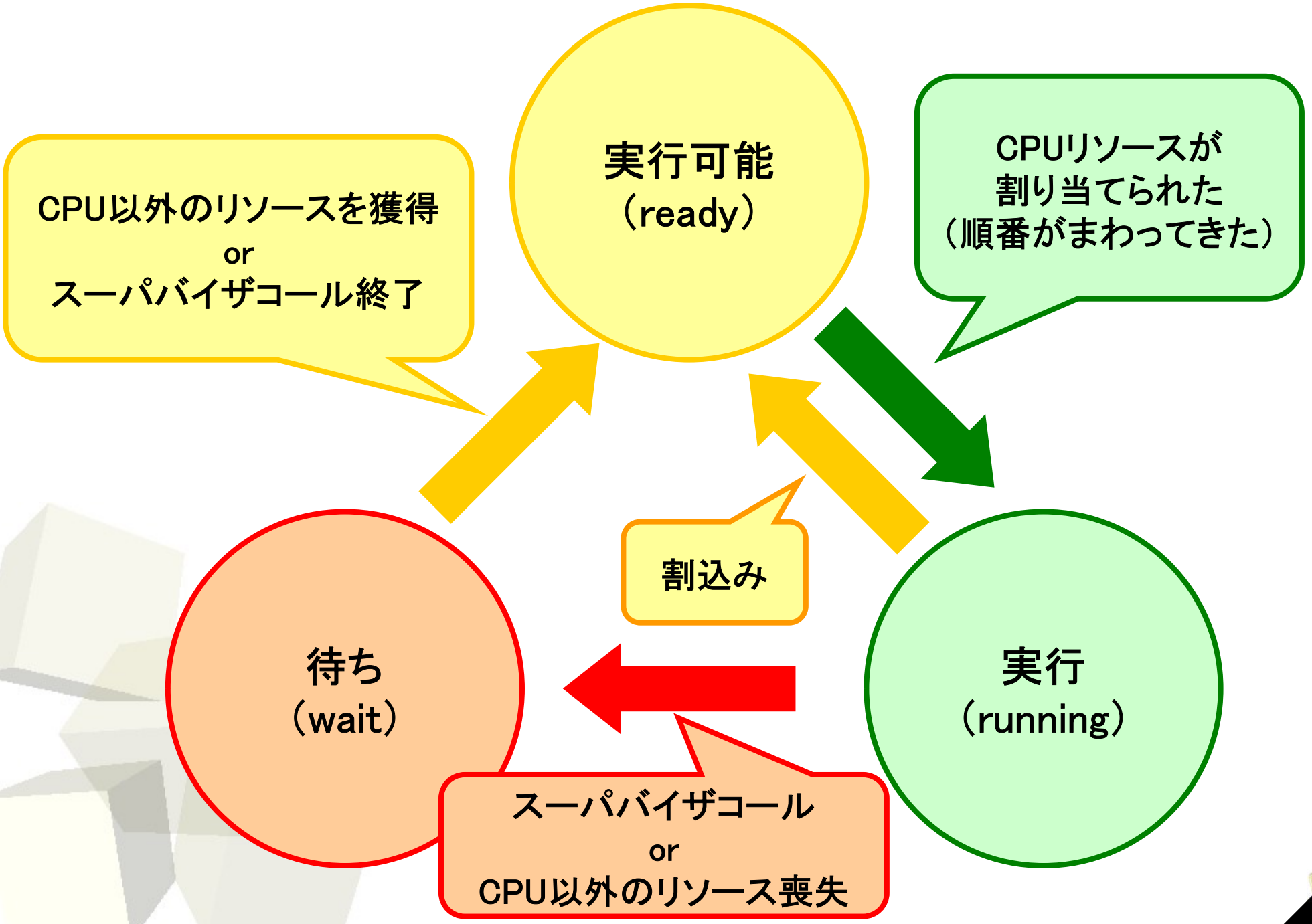
今日の内容

スケジューリング

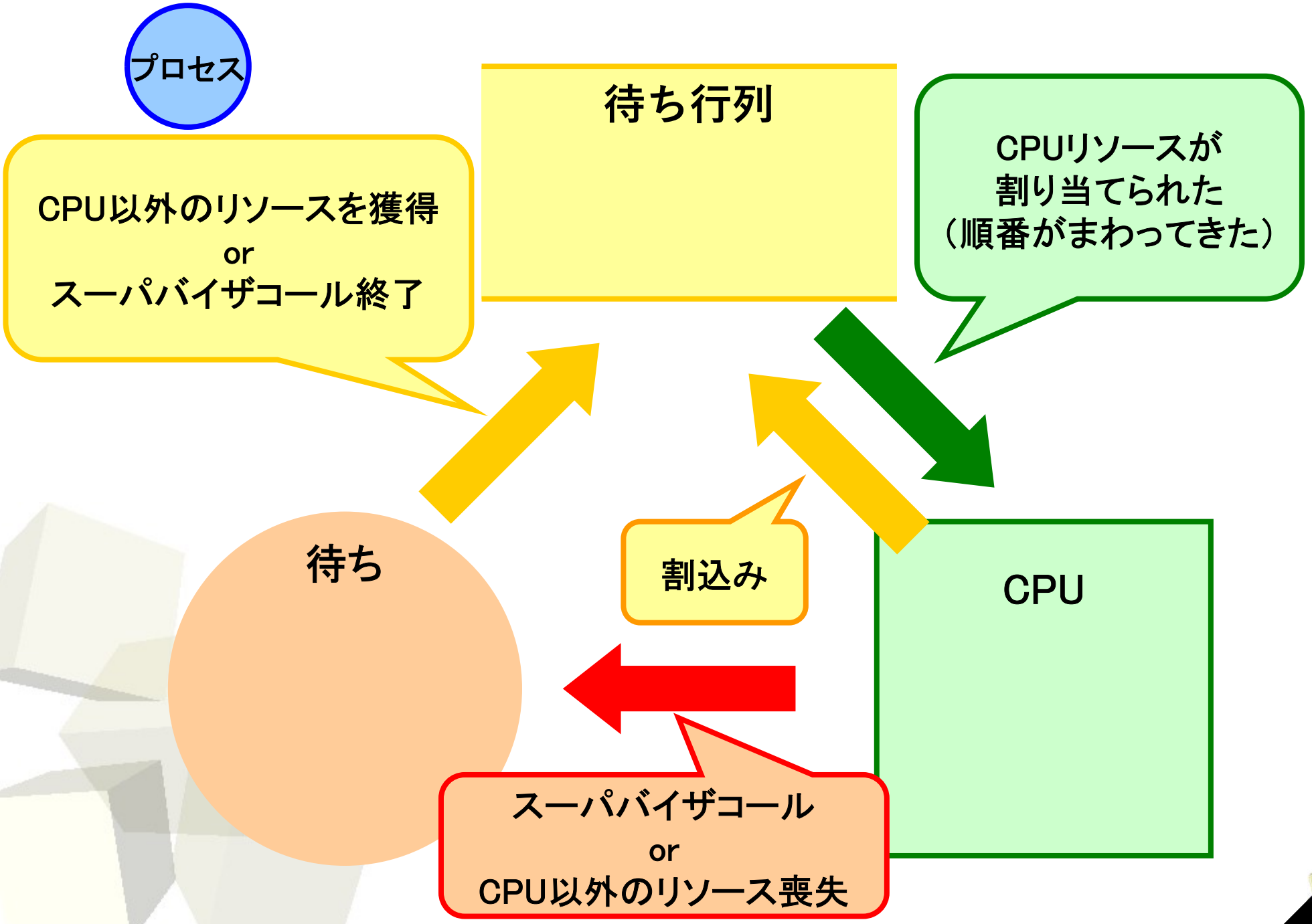
- スケジューリングの基本
- 様々なスケジューリング手法
- 実際のOSにおけるスケジューリング例

3.1
スケジューリングの基本

復習：プロセスの三状態


実行可能
（ready）
CPUリソースが
割り当てられた
（順番がまわってきた）
実行
（running）
割込み
スーパバイザコール
or
CPU以外のリソース喪失
待ち
（wait）
CPU以外のリソースを獲得
or
スーパバイザコール終了

プロセスの状態遷移


プロセス
CPU以外のリソースを獲得
or
スーパバイザコール終了
待ち行列
CPUリソースが
割り当てられた
（順番がまわってきた）
待ち
割込み
CPU
スーパバイザコール
or
CPU以外のリソース喪失

スケジューリング

- 実行プロセスの選択
  - CPUスケジューラが行う
  - 対話型処理では，数十～数百回/s
- スケジューリングアルゴリズム
  - 高速かつ軽量に行う必要
    - → オーバヘッド削減のため
- 基本
  - 待ち行列の先頭プロセスにCPUリソースを割り当て
    - → 全待ちプロセスの数に依存しない時間で
      スケジューリングが可能

プロセスの中断方式

- 実行プロセスの切り替えにはプロセスの中断が必要
  - 復習：CPU状態（PSW）の PCB への待避

- 中断方式
  - プリエンプション方式
    - → OSがプロセスから実行権を剥奪
    - → UNIX（LINUX), WindowsXP以降, MacOS X
  - ノンプリエンプション方式
    - → プロセスがOSに実行権を自主的に返還
    - → Windows 95, MacOS 9

プロセス暴走時には
システム停止も

3.2
スケジューリングの目的

スケジューリングの目的

- リソースを効率的に利用したい
  - CPUリソースは時分割により仮想化
    - → プロセス切り替えが多発
    - → 次に実行するプロセスを選択する機会も膨大
  - 切り替えごとにコスト(オーバヘッド)が発生
    - → スケジューリング次第で全体のオーバヘッドが増減
    - → 効率の悪いスケジューリング＝全体の性能低下
- 効率のよいスケジューリングが必要

効率化の指標

応答時間

- ある依頼した処理に対して
  応答が返ってくるまでに要する時間
  - → 対話処理：レスポンスタイム
    - 端末から入力した命令に対しシステムから結果を受け取るまでの時間
  - → バッチ処理：ターンアラウンドタイム
    - 投入したジョブに対しシステムから結果を受け取るまでの時間

スループット

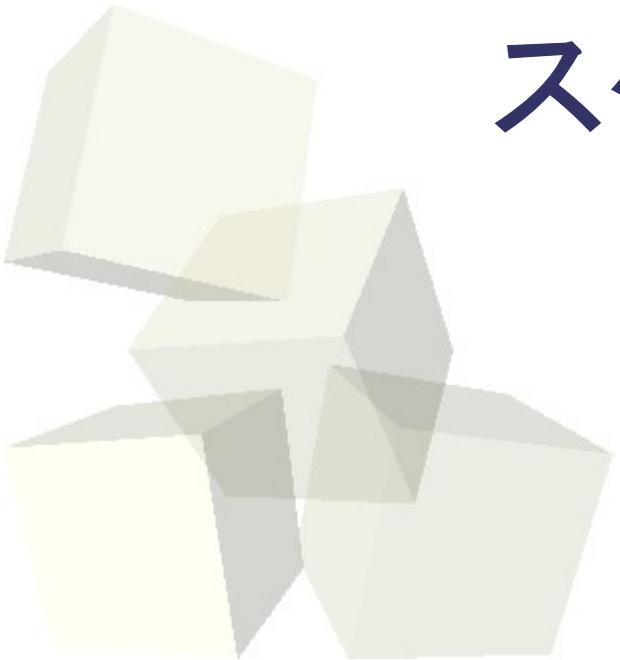
- ある単位時間においてシステムが処理する仕事量
  - → プロセス切り替えに必要となるオーバヘッド等は含まない
  - → ユーザにとって意義のある仕事をいかにこなせるか

効率化の指標

- 応答時間とスループットは
  トレードオフになる場合も
  - 例）
    - → 応答時間向上を追求
    - → 対話型処理を優先的に
    - → TSSのクオンタムを短く
    - → 切り替え回数増加，切り替えオーバヘッド増加
    - → スループット低下

- ユーザの要求やシステムの性質に応じて適切な
  指標・スケジューリングを用いることが重要

3.3
さまざまな
スケジューリング方式



さまざまなスケジューリング方式

- FIFO (First In First Out)
  - 到着順スケジューリング，FCFS
- SPTF (Shortest Processing Time First)
  - 処理時間順スケジューリング
- PS (Priority Scheduling)
  - 優先度順スケジューリング
- RR (Round Robin)
  - ラウンドロビン
- MLF (Multi-Level Feedback)
  - 多重フィードバック

さまざまなスケジューリング方式

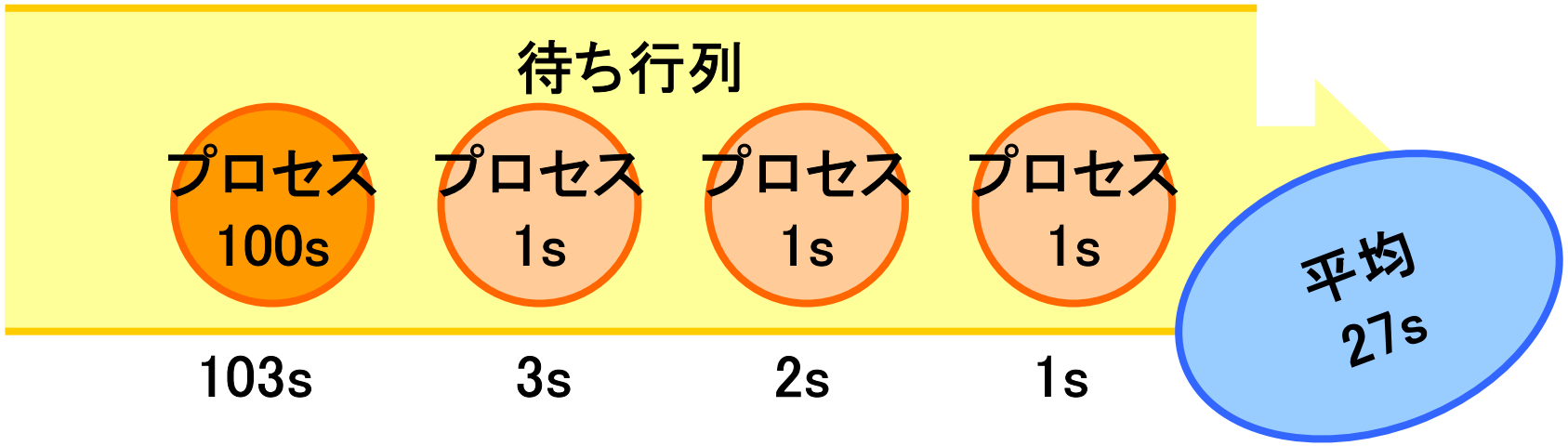
- FIFO (First In First Out)
  - 到着順スケジューリング，FCFS
- SPTF (Shortest Processing Time First)
  - 処理時間順スケジューリング
- PS (Priority Scheduling)
  - 優先度順スケジューリング
- RR (Round Robin)
  - ラウンドロビン
- MLF (Multi-Level Feedback)
  - 多重フィードバック

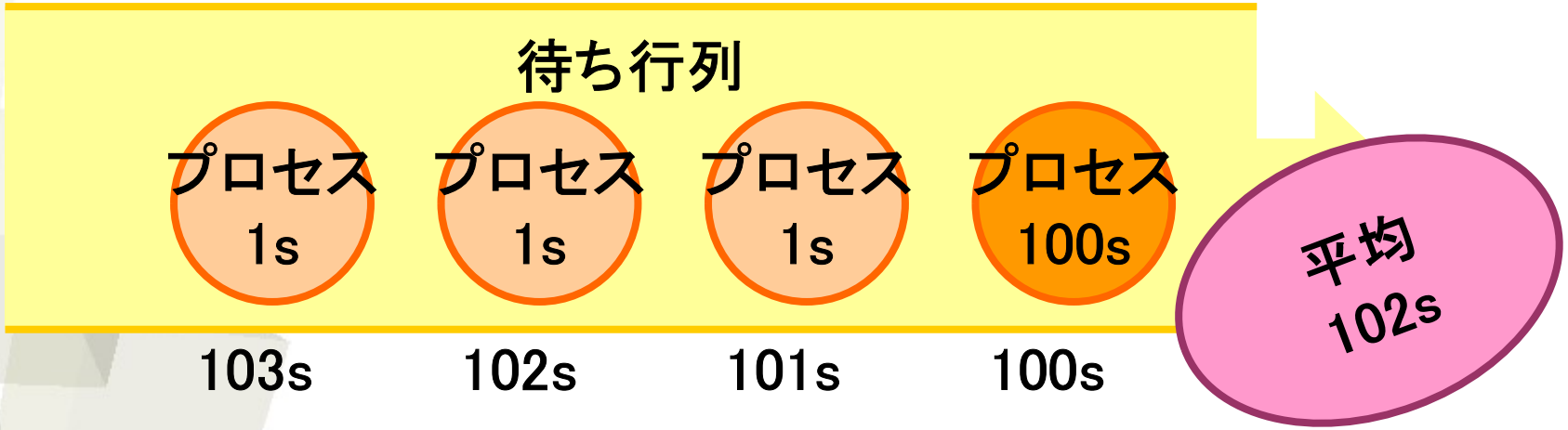
FIFO

- 到着順スケジューリング
  - FCFS: First Come First Served
  - FIFO: First In First Out
- 常に待ち行列の先頭から処理
  - ○単純
    - → プロセス選択機構も簡単になるし，選択オーバヘッドも小
  - ○公平
    - → 追い抜き禁止
  - ×ターンアラウンドタイムは良くない

FIFOの欠点

- ターンアラウンドタイム





さまざまなスケジューリング方式

- FIFO (First In First Out)
  - 到着順スケジューリング，FCFS
- SPTF (Shortest Processing Time First)
  - 処理時間順スケジューリング
- PS (Priority Scheduling)
  - 優先度順スケジューリング
- RR (Round Robin)
  - ラウンドロビン
- MLF (Multi-Level Feedback)
  - 多重フィードバック

SPTF

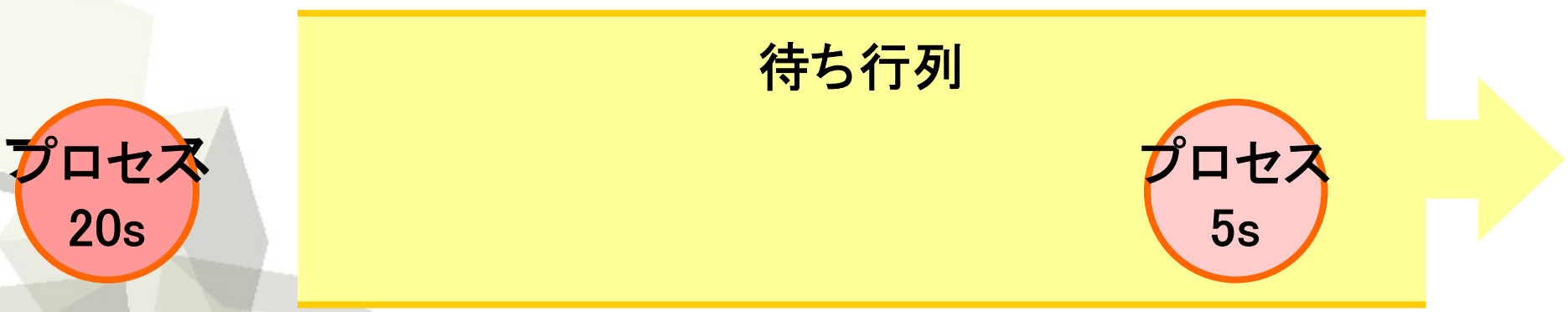
- FIFOの欠点は，
  各プロセスの「重さ」を考慮していないのが原因

処理時間順スケジューリング

- SPTF: Shortest Processing Time First
- 待ち行列内プロセスを処理時間順でソート
- 亜種：残り処理時間順（SRTF）

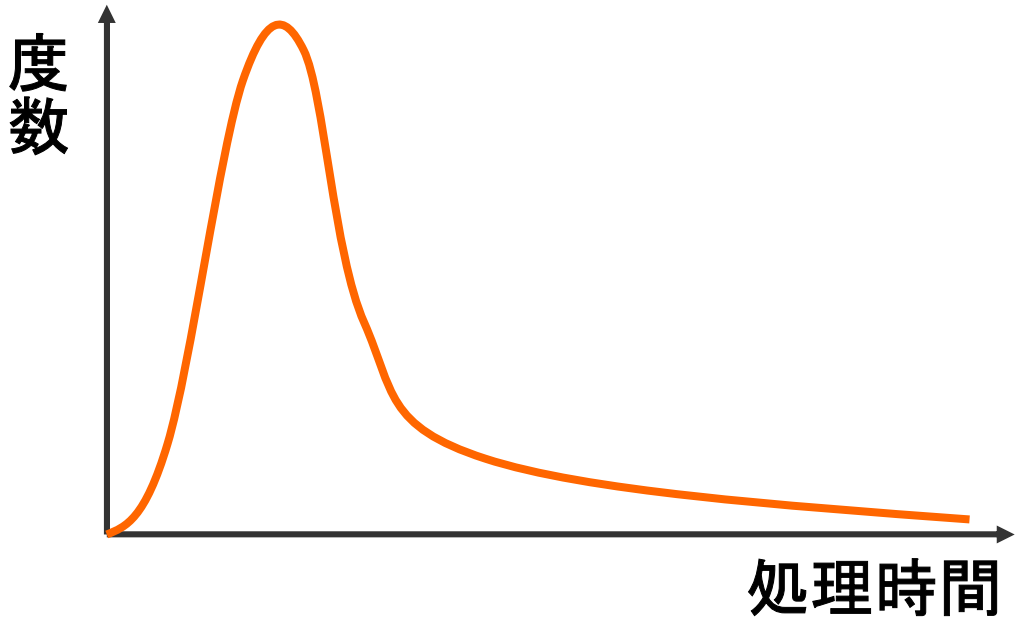
SPTF

- 処理時間の短いプロセスから順に処理
  - ○応答時間最短；理想的
  - ×実装不可能
    - → 各プロセスの処理時間を事前に知ることはできない



SPTFの実装手法

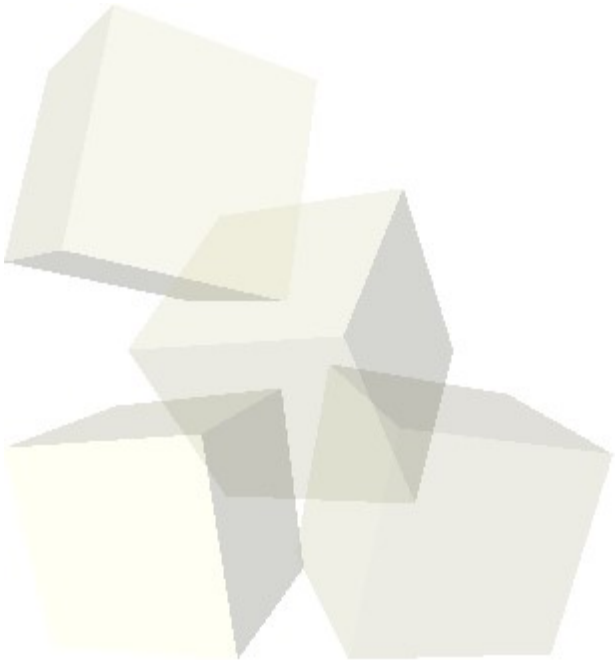
- プロセスの処理時間を推定してスケジューリング
  - 経験則（heuristic）から近似的に処理時間を求める
- 対話型処理のプロセス処理時間



  - ほとんどのプロセスは短時間で終了
  - 短時間で終了しないプロセスは，なかなか終了しない

SPTFの近似実装

- すでに実行した時間から
- 入出力処理から



さまざまなスケジューリング方式

- FIFO (First In First Out)
  - 到着順スケジューリング，FCFS
- SPTF (Shortest Processing Time First)
  - 処理時間順スケジューリング
- PS (Priority Scheduling)
  - 優先度順スケジューリング
- RR (Round Robin)
  - ラウンドロビン
- MLF (Multi-Level Feedback)
  - 多重フィードバック

■ 各プロセスに優先度を付加

- 静的優先度：プロセス生成時に指定した優先度を使用
  - → 例）・プロセスの種類ごとに優先度を規定
    リアルタイムプロセス ＞ OS ＞ 対話型 ＞ バッチ
- 動的優先度：プロセス実行中に優先度を適宜変化
  - → 例）・既実行時間に応じて優先度を変化
    ・入出力操作直後のプロセスの優先度を高く

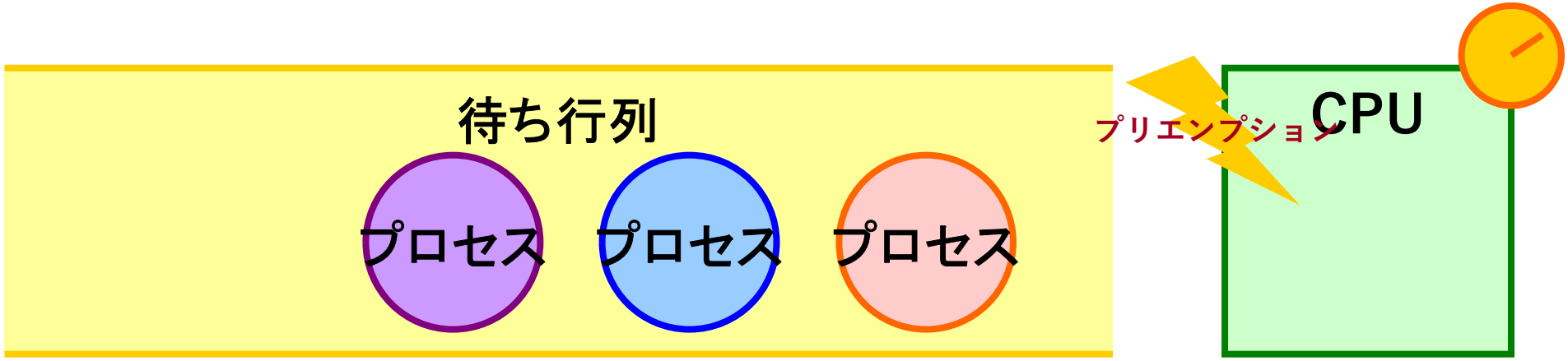
- ○優先度を適切に設定できれば非常に有効
- ×高負荷時，優先度の低いプロセスがなかなか実行権を獲得できない（starvation）
  - → 待ち時間に応じた優先度変化（aging）などで対処

さまざまなスケジューリング方式

- FIFO (First In First Out)
  - 到着順スケジューリング，FCFS
- SPTF (Shortest Processing Time First)
  - 処理時間順スケジューリング
- PS (Priority Scheduling)
  - 優先度順スケジューリング
- RR (Round Robin)
  - ラウンドロビン
- MLF (Multi-Level Feedback)
  - 多重フィードバック

RR

TSSで用いられる方式



- 待ち行列から選択されたプロセスに，微少なCPU利用時間（クオンタム）を割り当て

クオンタム

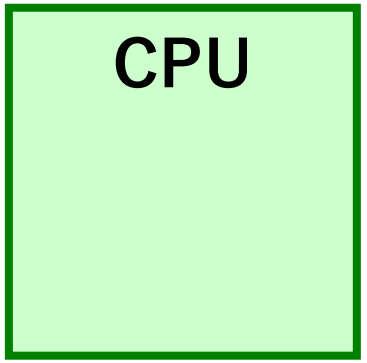
- クオンタム≒無限大：RR = FIFO
- クオンタム＝極小　：処理時間の短いプロセス有利

さまざまなスケジューリング方式

- FIFO (First In First Out)
  - 到着順スケジューリング，FCFS
- SPTF (Shortest Processing Time First)
  - 処理時間順スケジューリング
- PS (Priority Scheduling)
  - 優先度順スケジューリング
- RR (Round Robin)
  - ラウンドロビン
- MLF (Multi-Level Feedback)
  - 多重レベルフィードバック

MLF

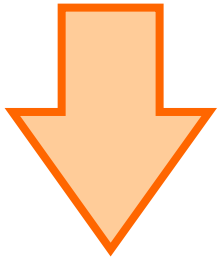
- Multi-Level Feedback （from Multics Project）
  - 優先度別に待ち行列を用意
  - プロセスは，クオンタムを得るごとに
    より優先度の低い待ち行列に移される



MLF

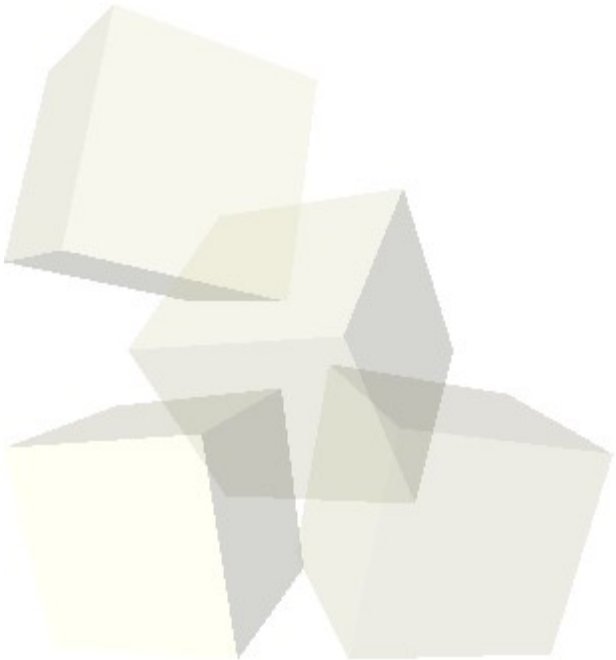
Multi-Level Feedback

- 複数のクオンタムを必要とするようなプロセス
  （すなわち長い時間がかかるプロセス）
  は，どんどん優先度が下がってゆく



- SPTFの良い近似になっている

3.4
スケジューリングアルゴリズムの実行例



FIFOの例

プロセス	処理時間 [s]	到着時刻 [s]
A	10	0
B	20	2
C	5	6

t=2　t=6　t=10　　t=30　t=35

平均
22.3s

SPTFの例

プロセス	処理時間 [s]	到着時刻 [s]
A	10	0
B	20	2
C	5	6

t=2　t=6　t=10　t=15　t=35

平均
17.3s

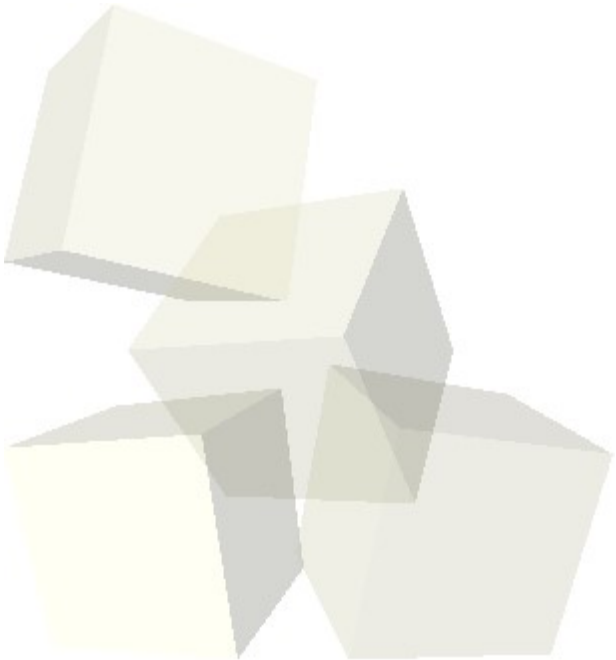
RRの例

プロセス	処理時間 [s]	到着時刻 [s]
A	10	0
B	20	2
C	5	6

t=2　t=6　t=20　t=23　t=35

平均
23.3s

3.5
事例：
UNIXにおける
スケジューリング



- 実際のOS (UNIX) の具体的実装例

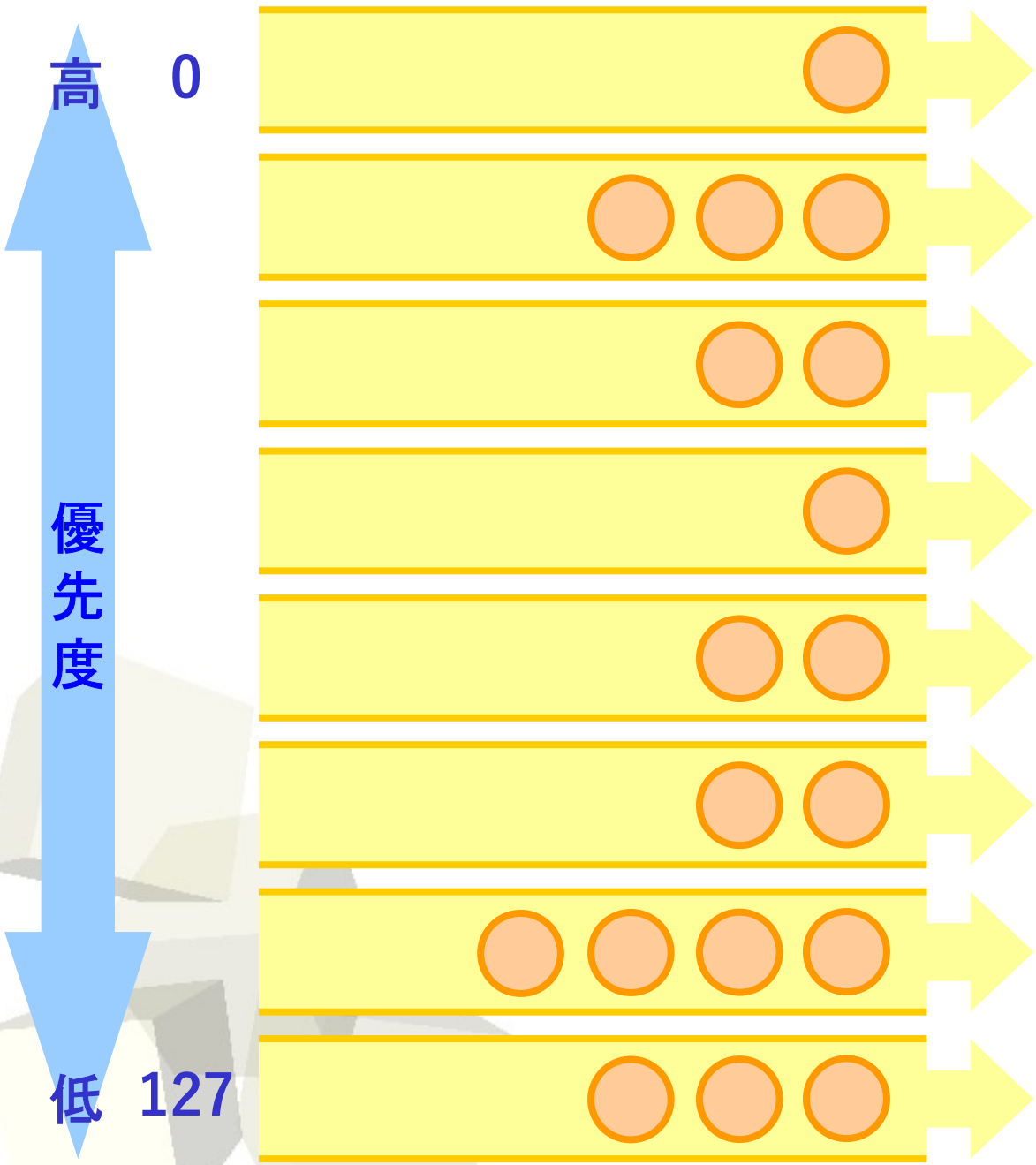
- UNIX SystemV
  - Solaris
  - AIX
  - HP-UX

- Linux 2.6.0 (2.6.22まで)

- 多重レベルフィードバックがベース

- 相違点
  - 実行の終わったプロセスは，同じ優先度の待ち行列に再登録
  - スーパバイザモードで実行されるプロセスは，そのプリエンプションの原因から優先度を決定
  - ユーザモードで実行されるプロセスは，以下で決定
    - → 静的優先度
    - → 動的優先度（過去のCPUリソースの割当状況から計算）
    - → NICE値（ユーザが指定する優先度）

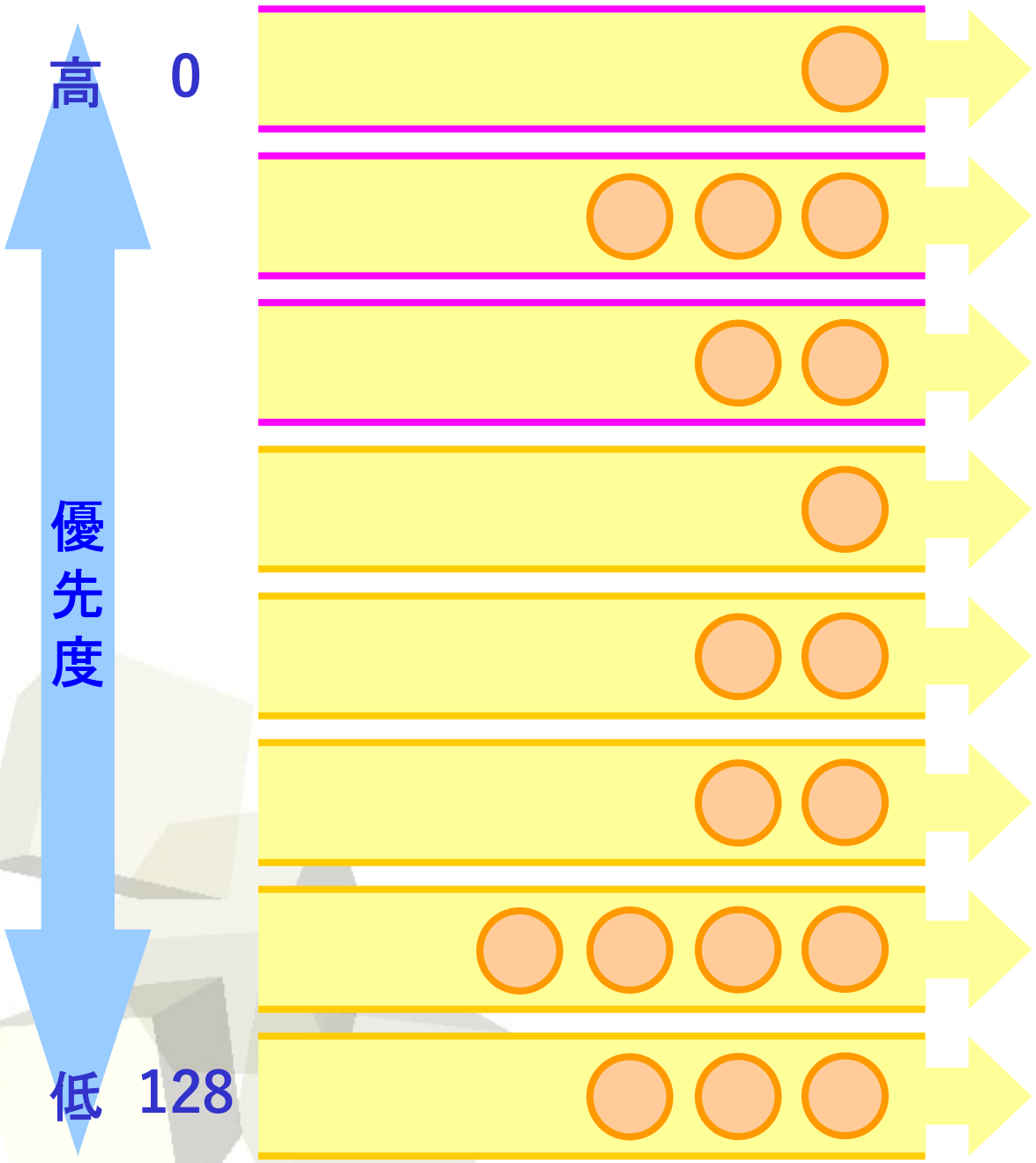
SysVのスケジューリング



例）ディスク入出力待ち

他のリソースを確保している可能性が高く，早く実行してやらないと他プロセスを足止めする。

SysVのスケジューリング

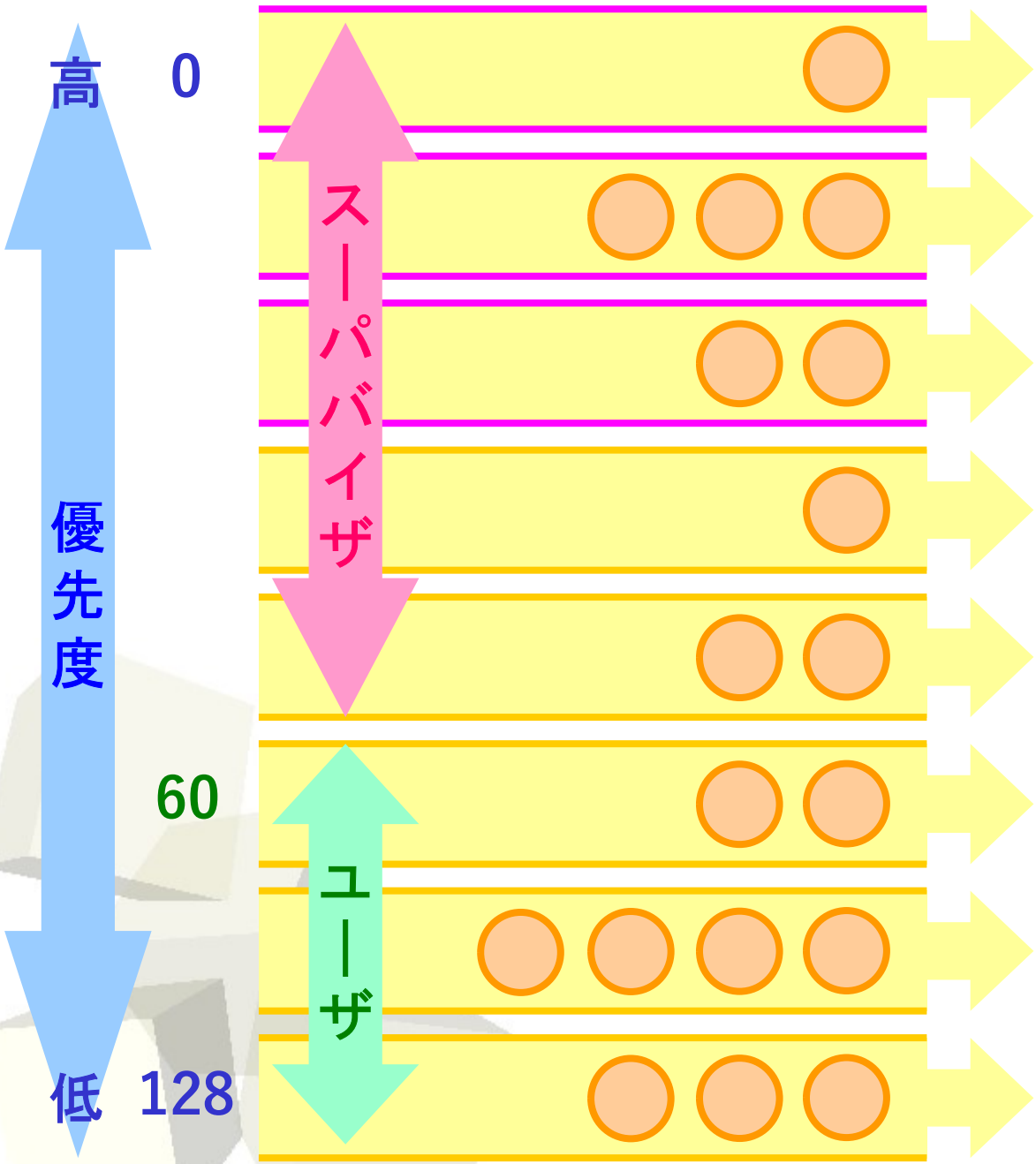


ユーザモード優先度

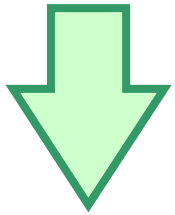
- P_USER
  + NICE
  + P_CPU

- P_USER
  基準値 (=60)
- NICE
  ユーザ指定優先度
- P_CPU
  動的優先度

SysVのスケジューリング

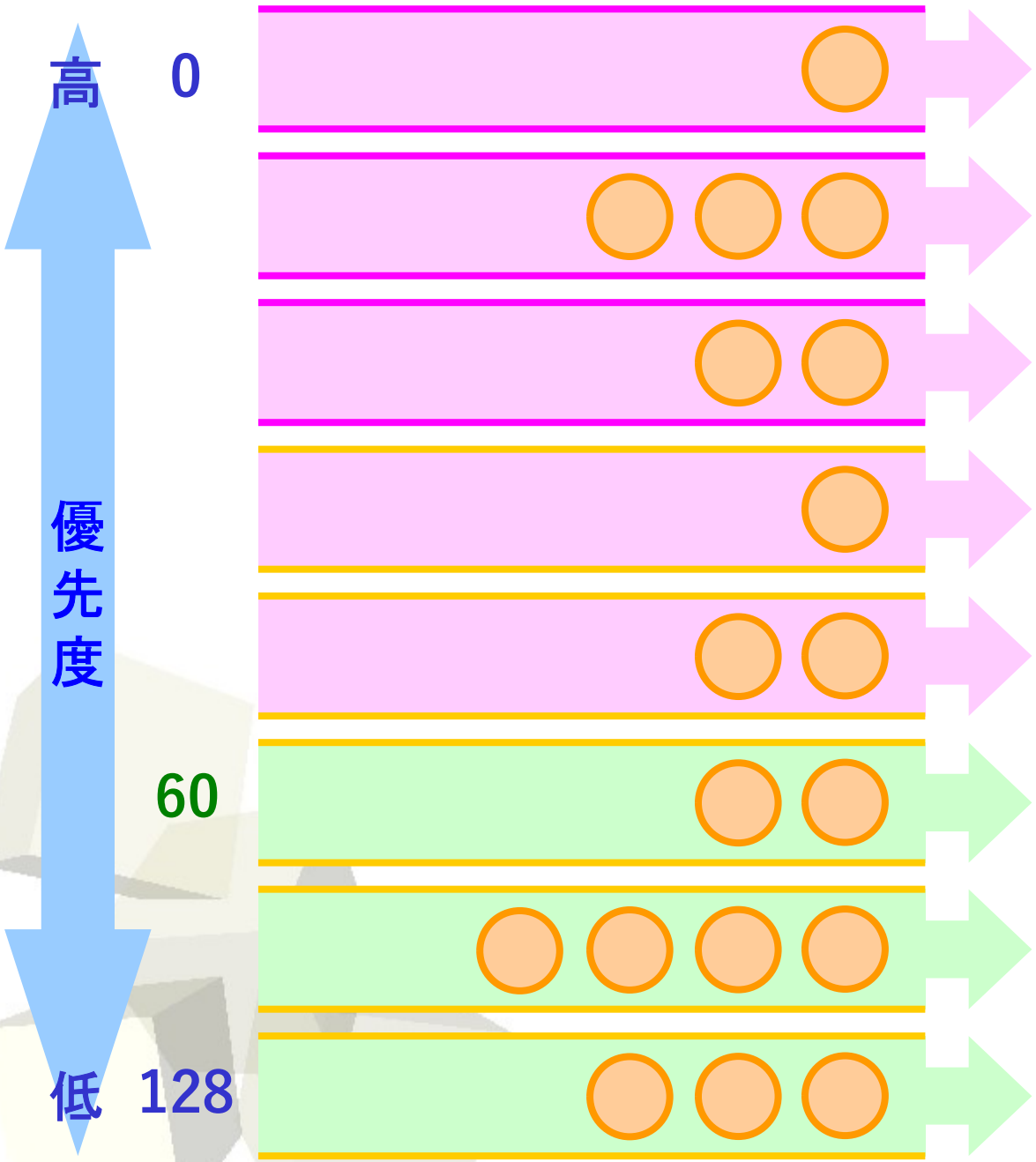


- P_USER
  - ユーザモードプロセスの基準値
  - 必ずこの値よりも優先度が低くなる



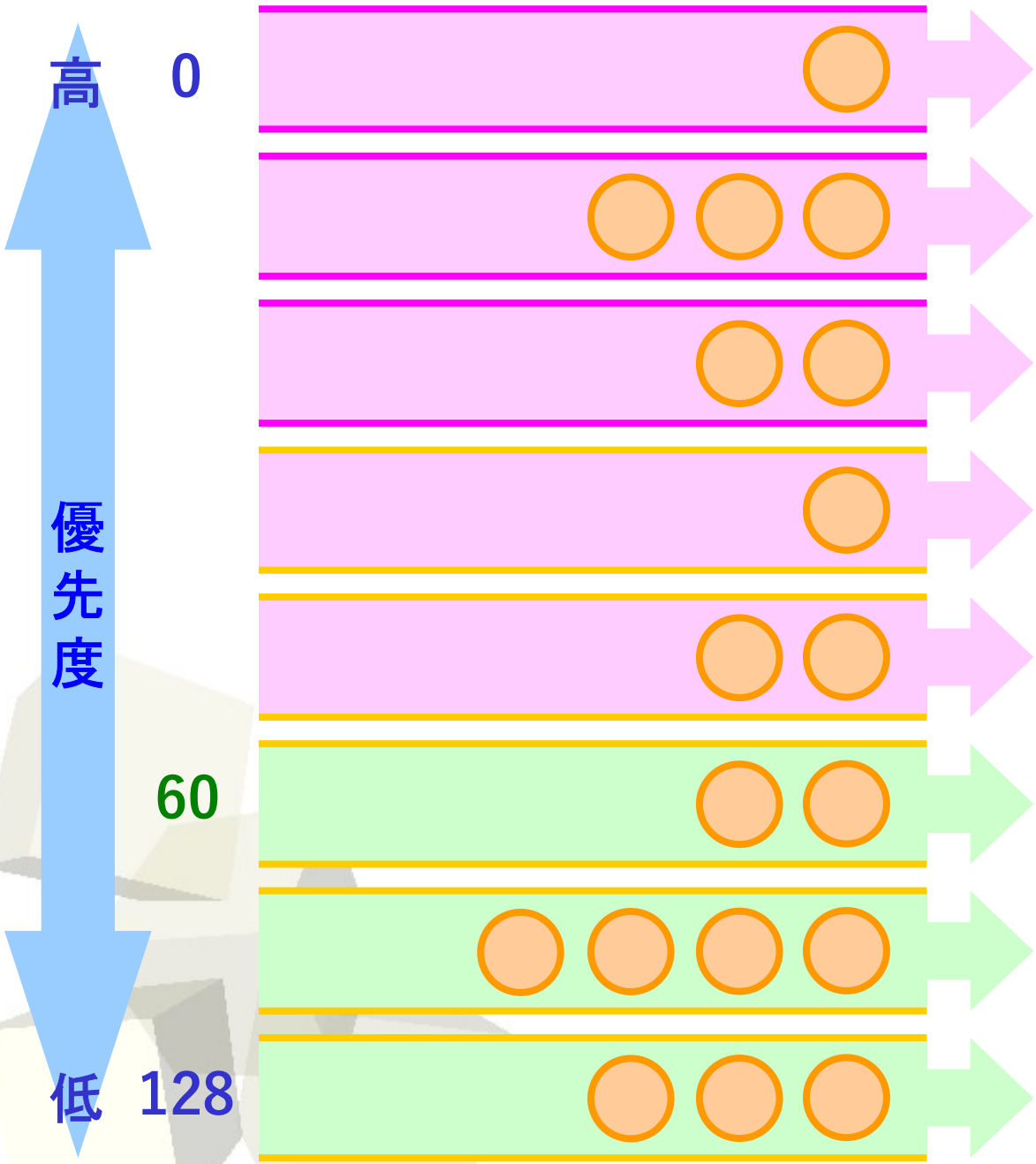
  - スーパバイザモードプロセスとの境界
  - 通常 60

SysVのスケジューリング



- NICE
  - ユーザが指定する優先度
  - ユーザは実行するプロセスの優先度を下げることができる
  - （基本的には）上げることはできない

SysVのスケジューリング



- P_CPU
  - LOAD
    - → 過去にどれだけCPUを割り当てられたか
  - P_CPU
    - → 前のP_CPU値

- $P\_CPU_t = \frac{1}{2}(P\_CPU_{t-1} + LOAD_{t-1})$

P_CPU

- LOAD値と過去のP_CPU値を
  一定の割合で結合

- LOAD値
  - 直前のCPU割当状況から計算
  - 変動：大
- P_CPU値
  - 過去の履歴を反映
  - 変動：緩やか

- 実際のOS (UNIX) の具体的実装例

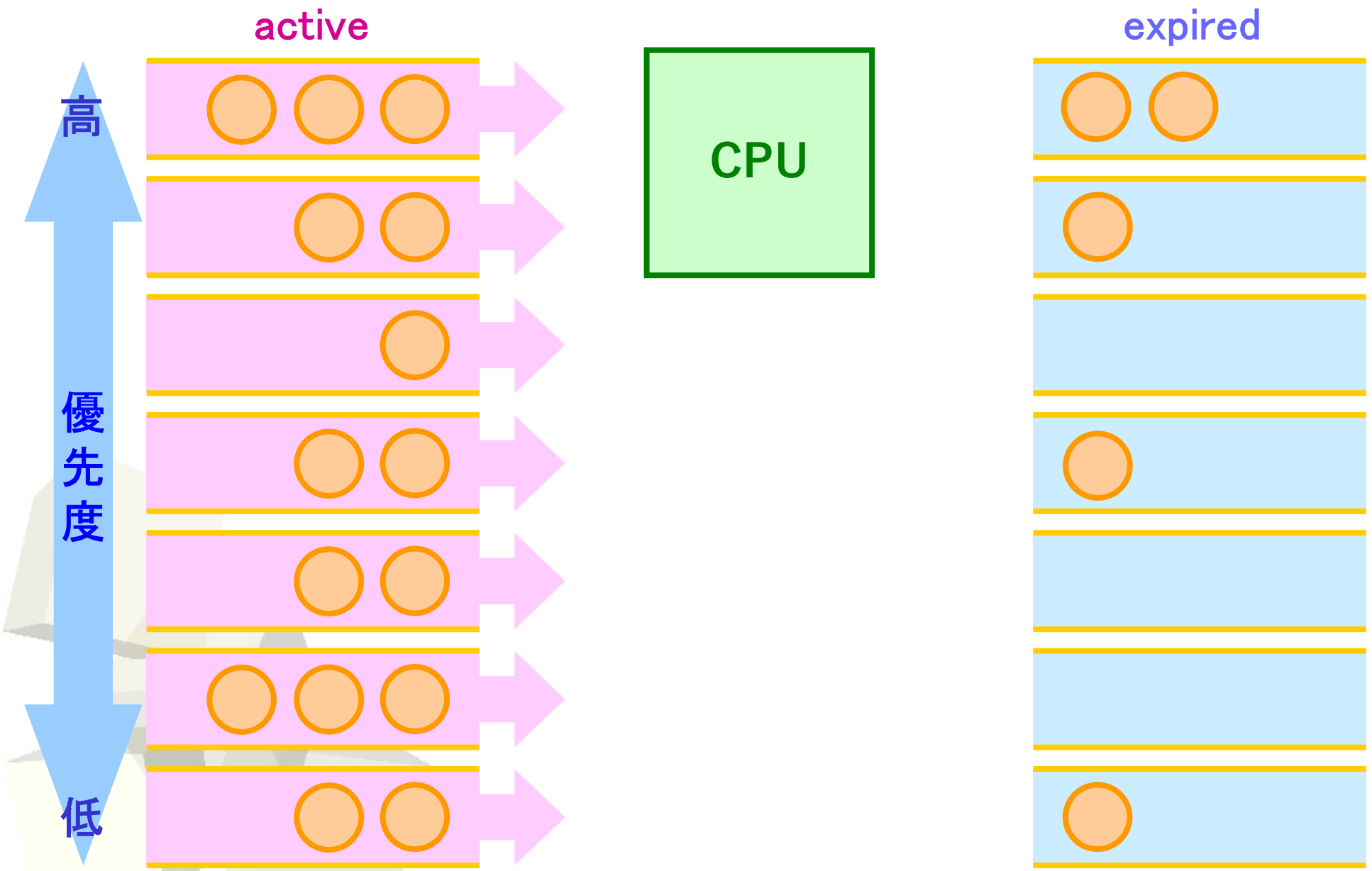
- UNIX SystemV
  - Solaris
  - AIX
  - HP-UX

- Linux 2.6

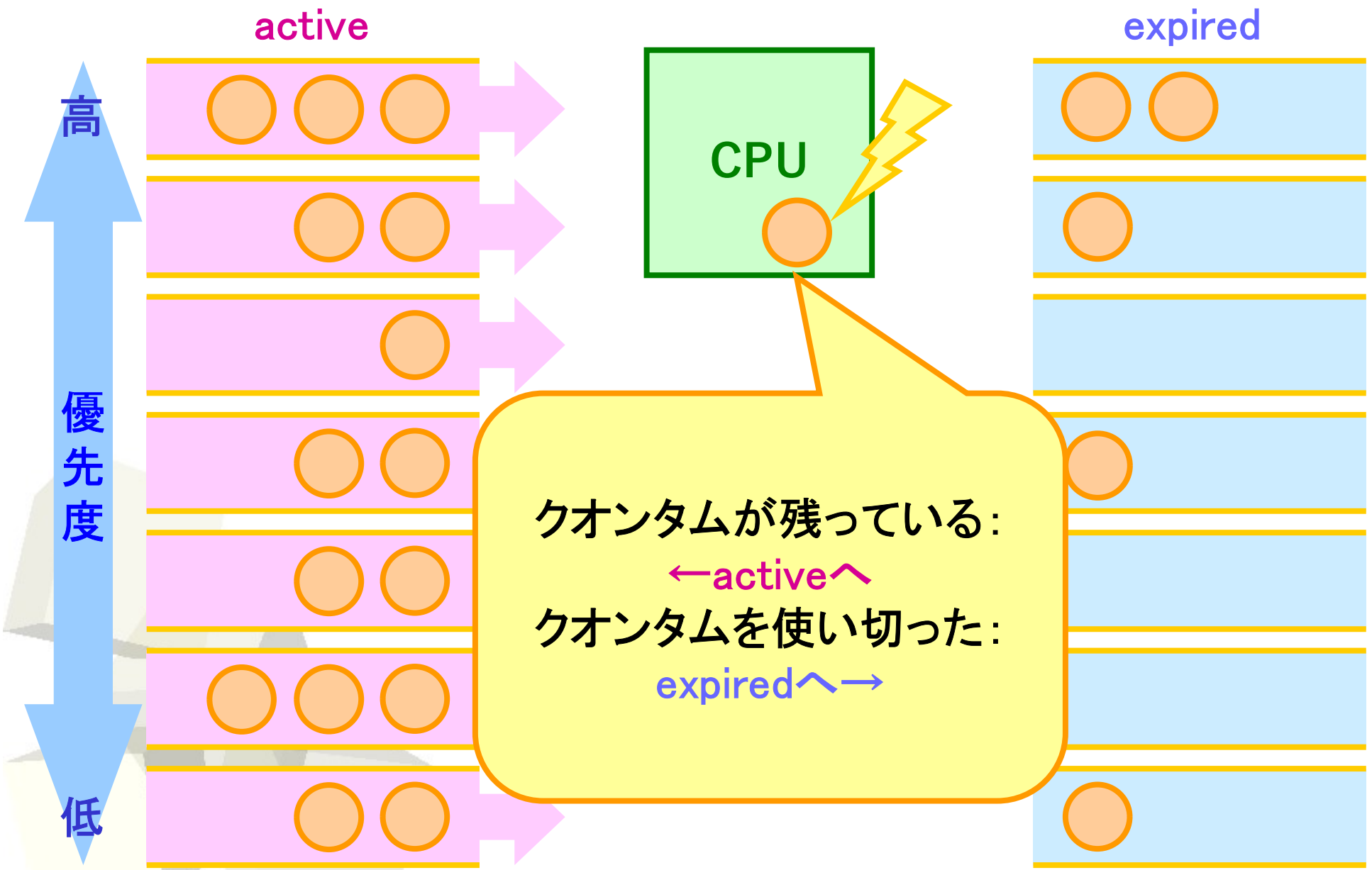
O(1)スケジューリング

- オーダ１
  - 問題サイズに関わらず，一定時間で処理できる
  - 参考）O(n)：オーダ ｎ
    問題サイズに比例して要する時間が増加：
    例）待ちプロセス数が2倍になると，
    　　プロセス決定に要する時間も2倍

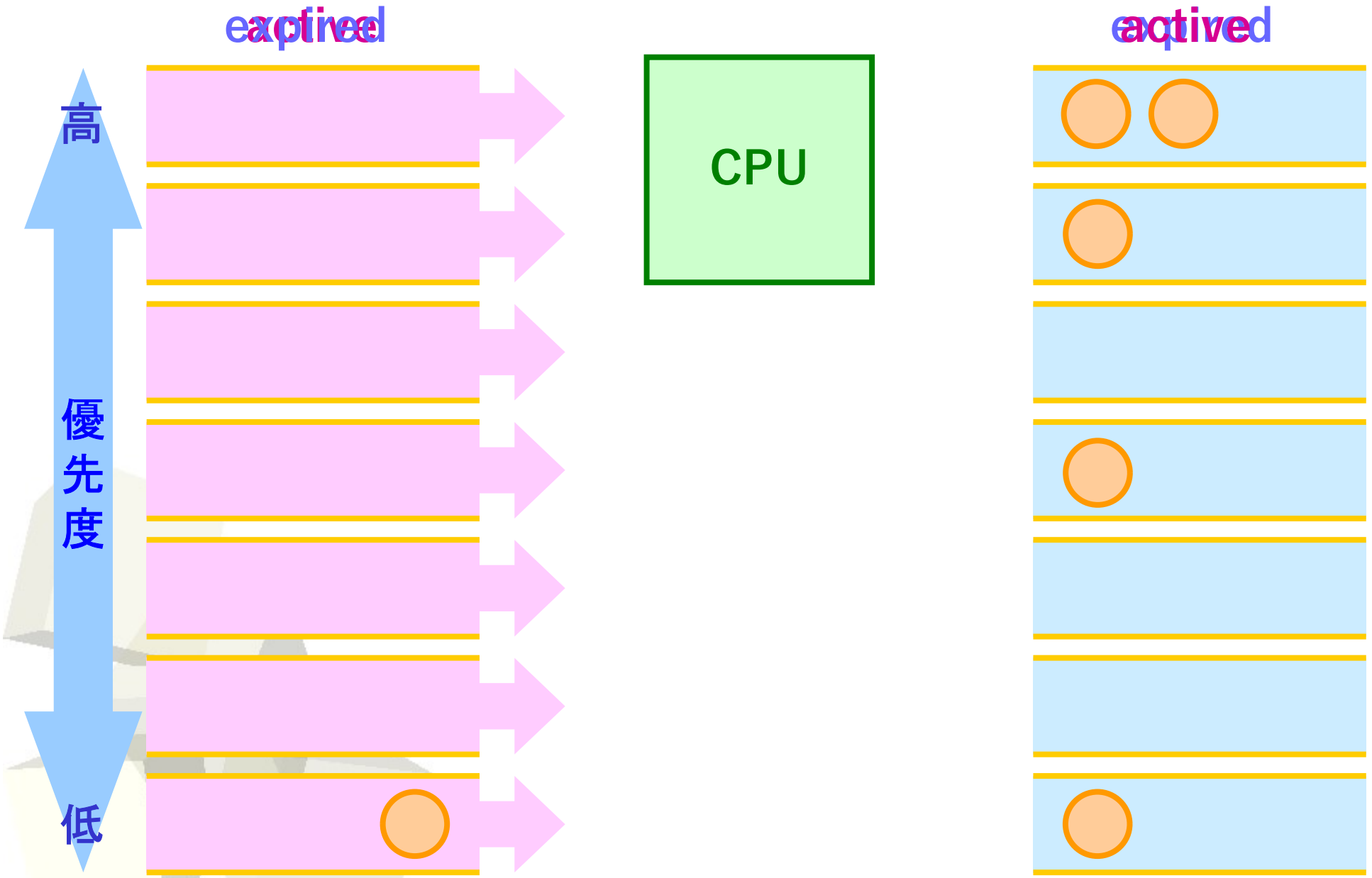
Linux 2.6 のスケジューリング



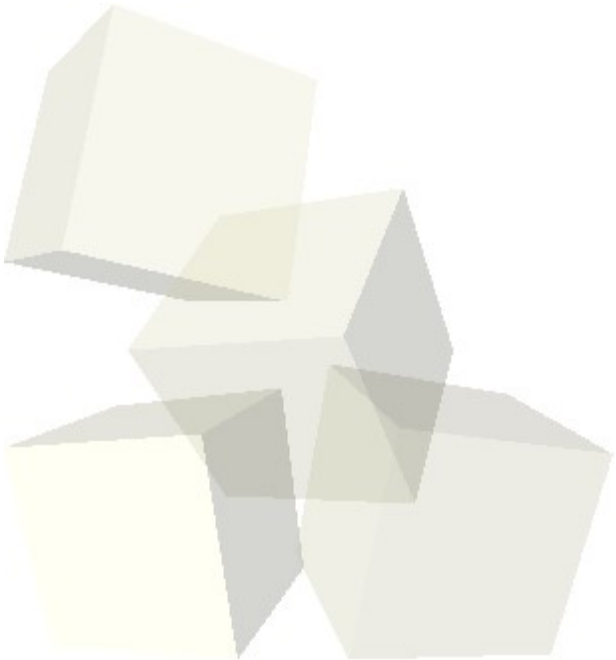
Linux 2.6 のスケジューリング



Linux 2.6 のスケジューリング



まとめ

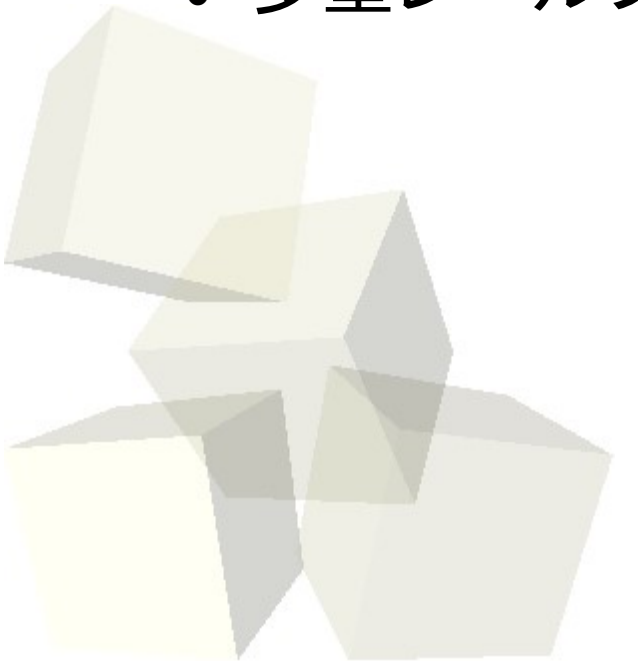


まとめ：スケジューリング方式

- FIFO (First In First Out)
  - 到着順スケジューリング，FCFS
- SPTF (Shortest Processing Time First)
  - 処理時間順スケジューリング
- PS (Priority Scheduling)
  - 優先度順スケジューリング
- RR (Round Robin)
  - ラウンドロビン
- MLF (Multi-Level Feedback)
  - 多重フィードバック

まとめ：処理時間順

- 処理時間順スケジューリング
  - 理論上，応答時間を最小にできる
  - 実装が不可能
- 近似により処理時間順を実現
  - 多重レベルフィードバック



まとめ:プロセス優先度

- 静的優先度
  - プロセス生成時に決定
- 動的優先度
  - 実行中に変化

- 優先度スケジューリング
  - 優先度の低いプロセスがstarvationに陥る可能性
  - aging等で解決

さらなる発展のため

- Multics
  - Wikipedia かなり説明は難しいので注意が必要
  - ggr“OS誕生からLinuxまでの歴史”
- プリエンプション（Wikipedia）
- Windows 3.xのマルチタスク（ggr “windows3.1 マルチタスク”）
  - https://www.atmarkit.co.jp/fwin2k/special/win9xorwin2k/column_winmultitask.html
  - Microsoft Windows 3.X (Wikipedia)