Real-Time Java
～組み込みから エンタープライズまで～

サン・マイクロシステムズ株式会社
ソフトウェア OEM 営業本部
草薙 昭彦

本日のトピック

- リアルタイムシステムとは
- Java リアルタイム仕様とは
- Sun Java Real-Time System 2.0

リアルタイム Java @ JavaOne 2007

- Java Playground
  - > Slot Car Challenge II
  - > ABB Industrial Robot
  - > Helicopter
  - > Trading Demo
- My Keynote
  - > ABB, Helicopter
- Several talks and BOFs

リアルタイム Java @ JavaOne 2007

Fastest Industrial Robot

Being assembled in real-time...

リアルタイムシステムとは

リアルタイムシステムとは

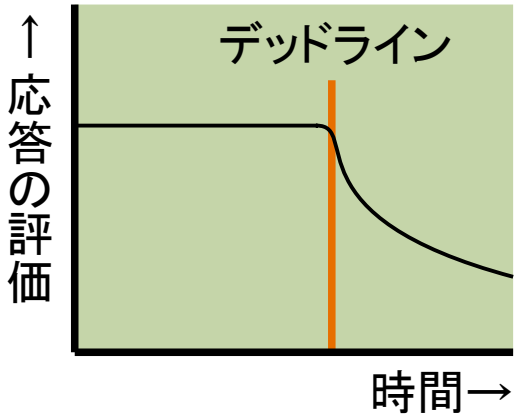
- リアルタイムシステム≠高速
- 予測可能な時間内に処理を行う

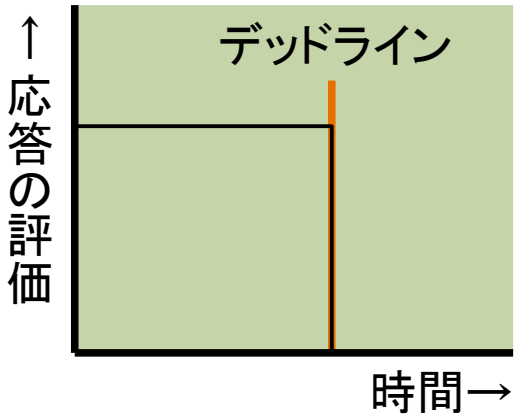
到着するリクエストに対して逐次応答するのではなく、重要なイベントとそうでないイベントを区別する

最も重要なイベントを一番に処理を行い、その他のイベントは可能になったときに実行する

リアルタイム要件

- ソフトリアルタイム
  - 応答のデッドラインをミスすると障害とみなされるが、処理は継続するアプリケーション
  - 例：ユーザーインターフェース
- ハードリアルタイム
  - デッドラインのミスがシステム停止に直結するアプリケーション
  - 例：自動操縦装置



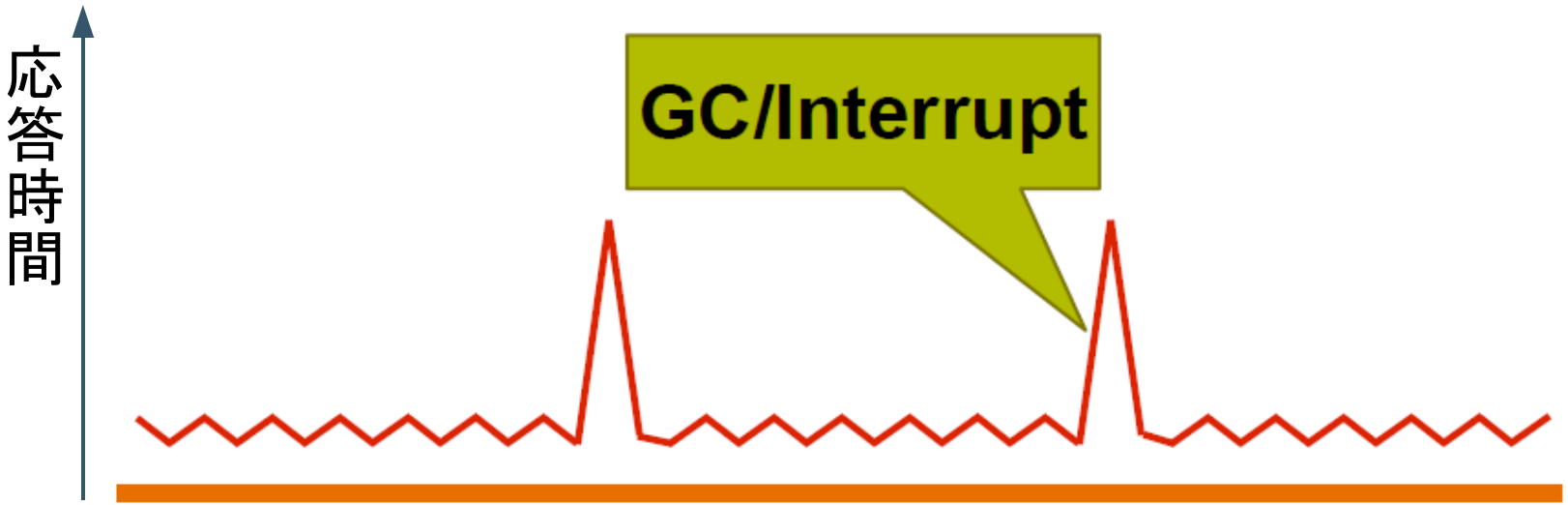


Java がリアルタイムシステムに使われない理由

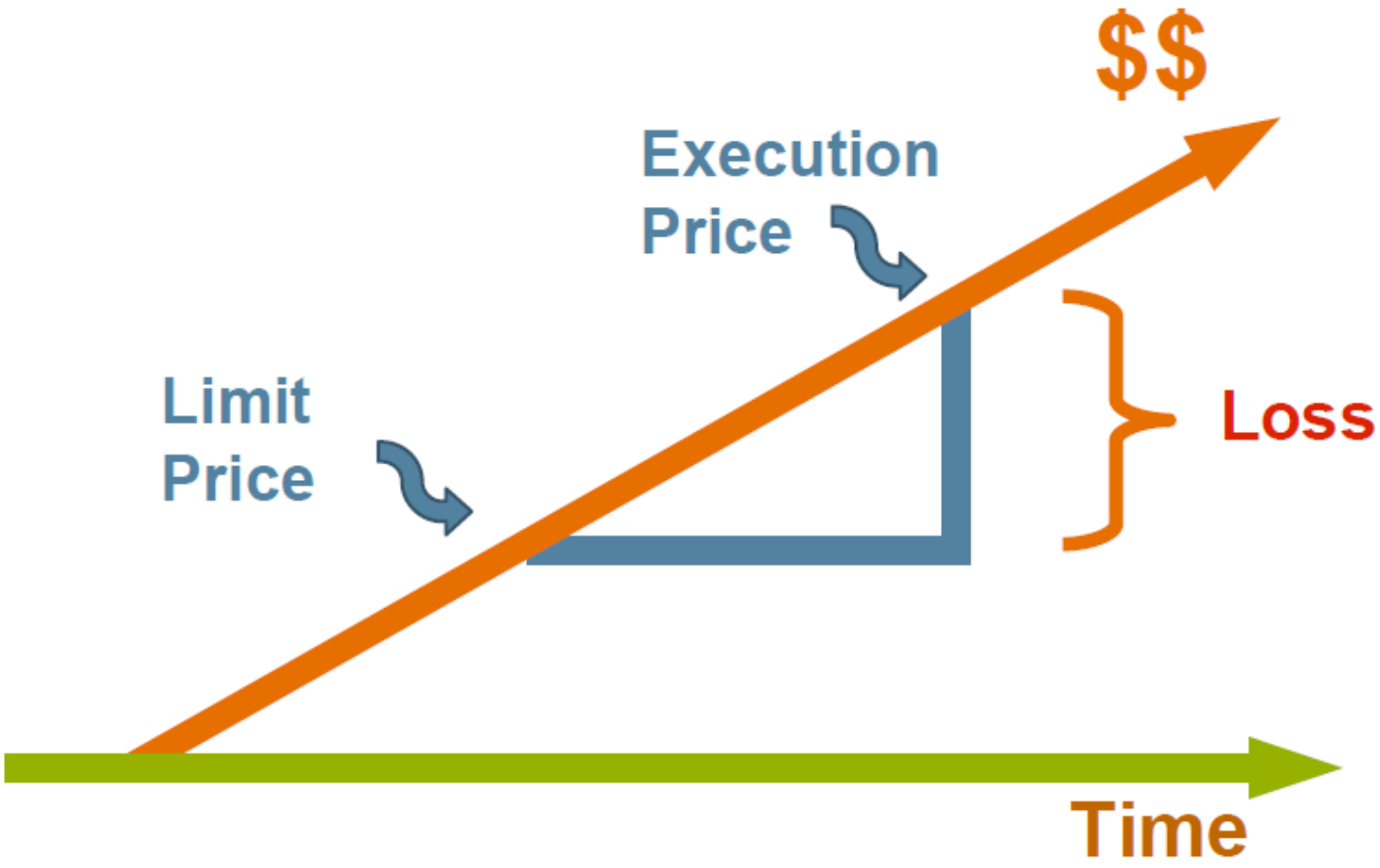
- 処理時間（応答時間）が予測できないから！
- 予測困難性を生み出す原因の例
  - ガベージコレクション
  - スレッドスケジューリング
  - 動的クラスローディング
  - JIT コンパイル

問題の例

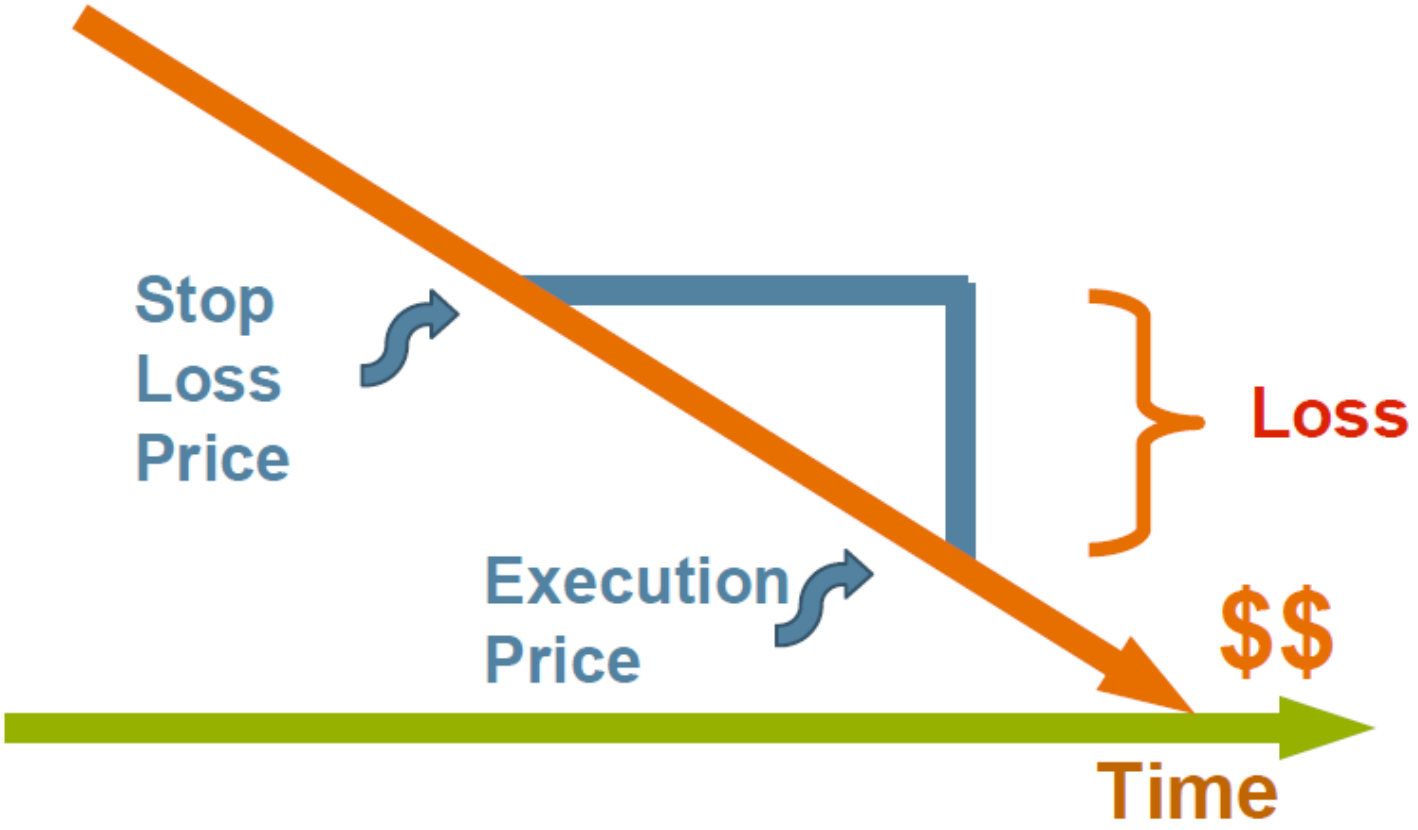
- Java プラットフォームの GC 、他のアプリケーションやスレッド、システム割り込みがトレーディングに遅延をもたらしてしまう



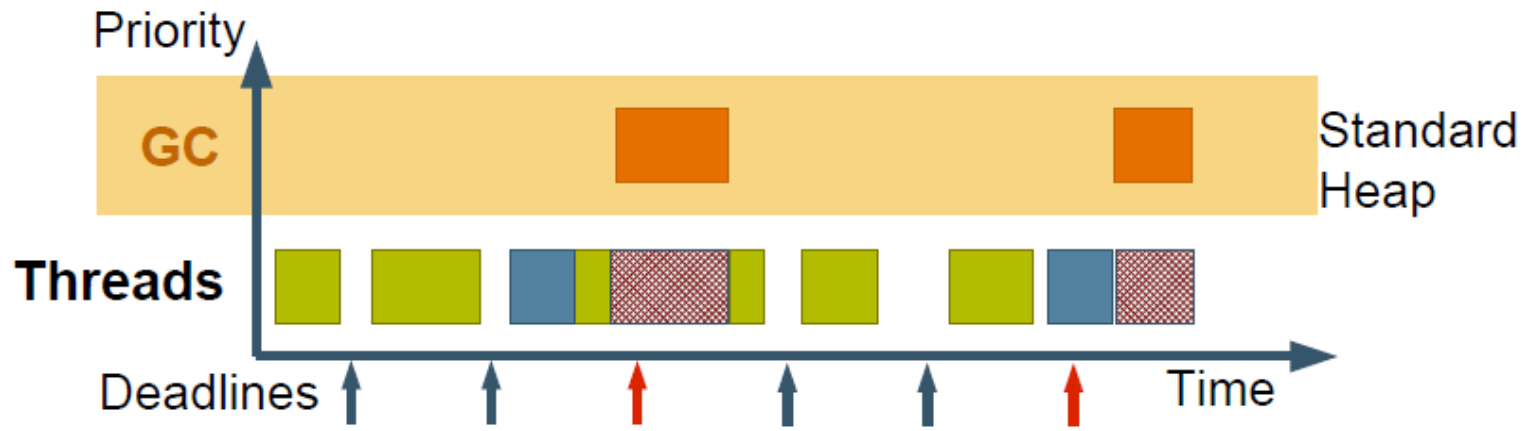
その結果 —— 指値注文



その結果 —— 逆指値・損切り注文



GC の一時停止は予測できない



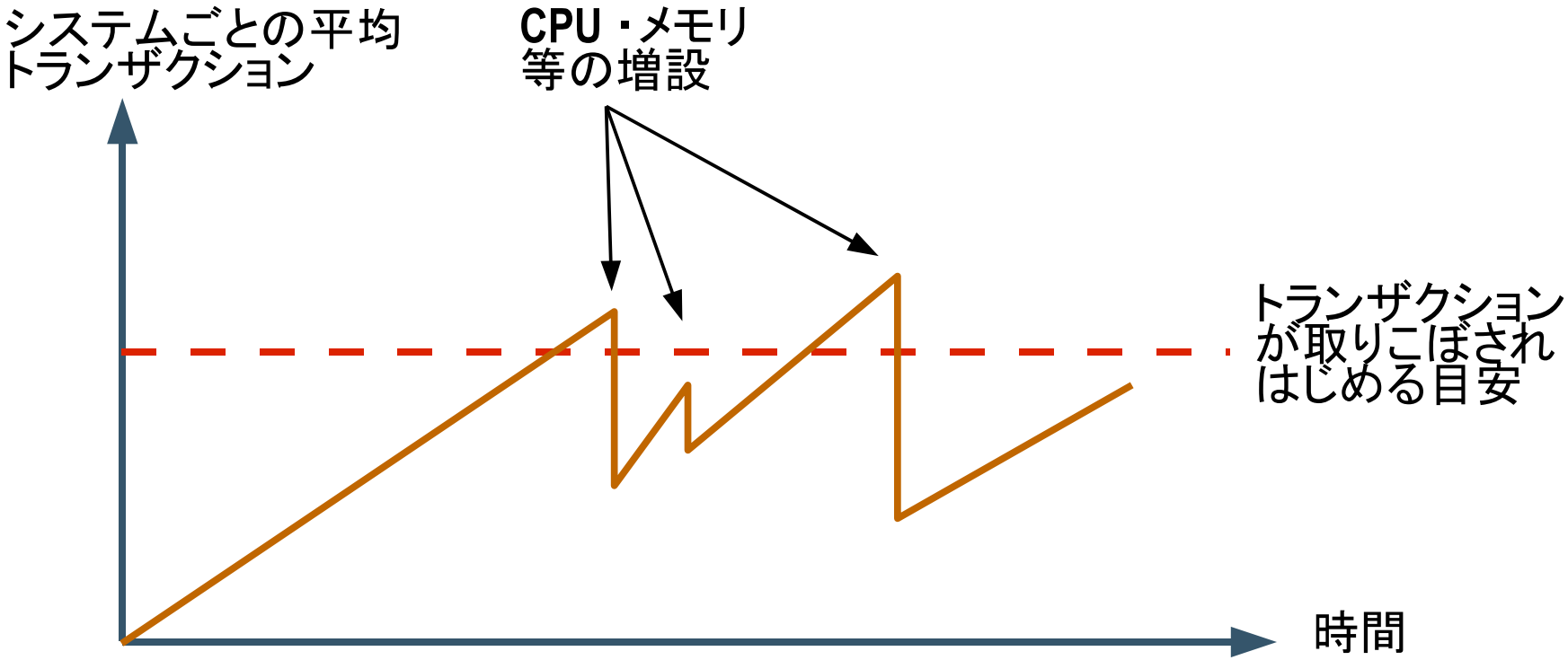
- 通常処理のスレッド
- 予期しないイベント
- GC によるスレッドの一時停止

ガベージコレクションは予期しないタイミングで起こり得る――デッドラインのミスを引き起こす

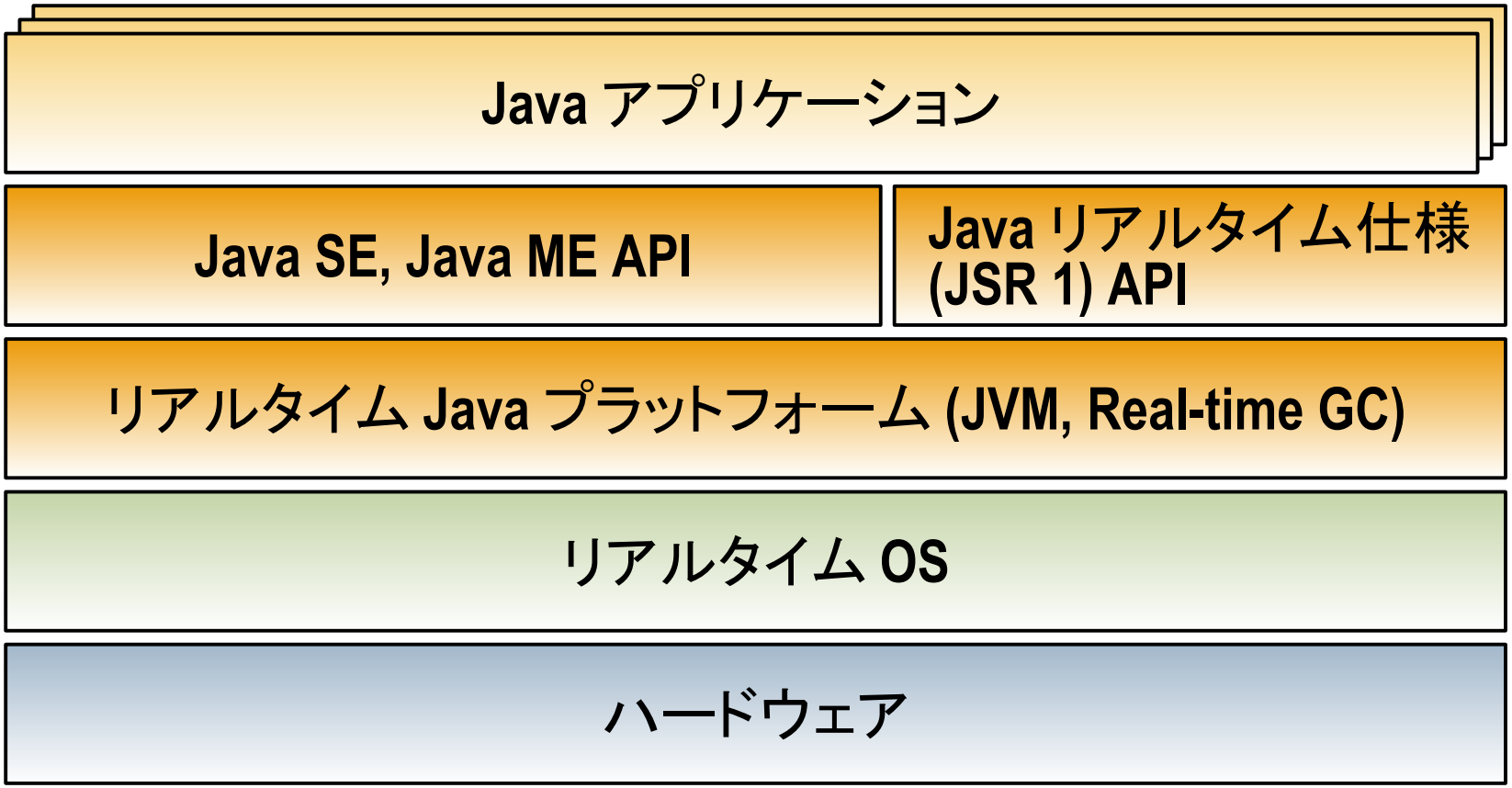
ガベージコレクションを正確に予測することは非常に難しい――が、最悪のタイミングで起こることは予測できる

システムのキャパシティプランニング

- 通常 40~60% 程度の空き容量で運用される
- トランザクションが増えてきた場合、どの時点でどれだけ増強すべきかの判断が難しい



リアルタイム Java 環境の構成



Java リアルタイム仕様とは

Java リアルタイム仕様の概要

- The Real-Time Specification for Java (RTSJ)
  JSR 001 (1.0), JSR 282 (1.1)
  - リアルタイムの振る舞いを Java でどう扱うかということを定義している標準規格
- 特徴
  - 全ての問題に効く特効薬ではないが、解決に役立つ強力なツール
  - 高い抽象化レベル、移植容易性、コスト・デッドラインといった概念の導入
  - 100% Java テクノロジー
  - 多くの専門家の参加により策定

RTSJ の機能

- スレッドスケジューリング
- メモリ管理
- 同期とリソース共有
- 非同期イベントのハンドリング
- 非同期制御転送
- 物理メモリアクセス
- 高精度クロック

HelloWorld

```
import javax.realtime.*;

public class HelloWorld {
  public static void main(String[] args) {
    RealtimeThread rt = new RealtimeThread() {
      public void run() {
        System.out.println("Hello World");
      }
    };
    rt.start();
  }
}
```

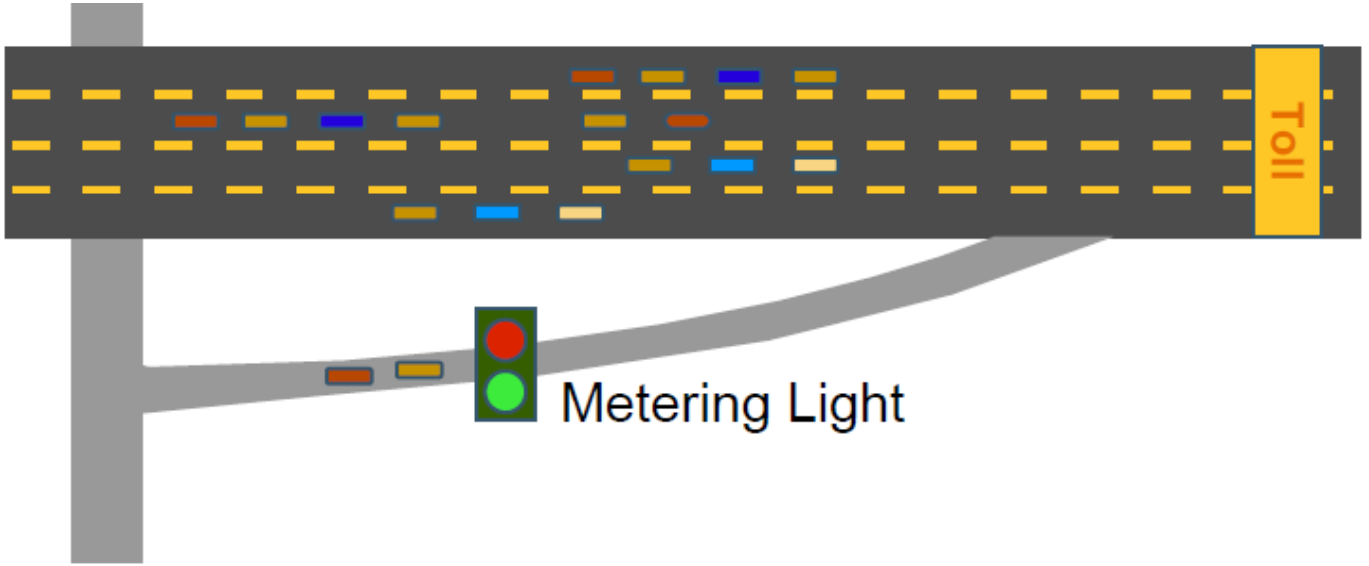
スレッドの拡張

- リアルタイム動作のベースとなるスレッド
- RTSJ で定義されているスレッドクラス
  - RealtimeThread
    - 細かい優先度の設定、定期的な動作、動作期限の設定、メモリモデルの指定などをサポート
    - ソフトリアルタイム
  - NoHeapRealtimeThread (NHRT)
    - ヒープにアクセスできない特殊な RealtimeThread
    - ハードリアルタイム

厳密なスレッド優先度

- 標準 Java
  - ラウンドロビンによるプリエンプティブスケジューリング
  - スケジューリングの優先度は保障されない
- RTSJ
  - 厳密な 28 の優先度に基づくプリエンプティブスケジューリング
  - 優先度継承プロトコルによる優先度逆転問題への対応

アナロジー:コミューターハイウェイ

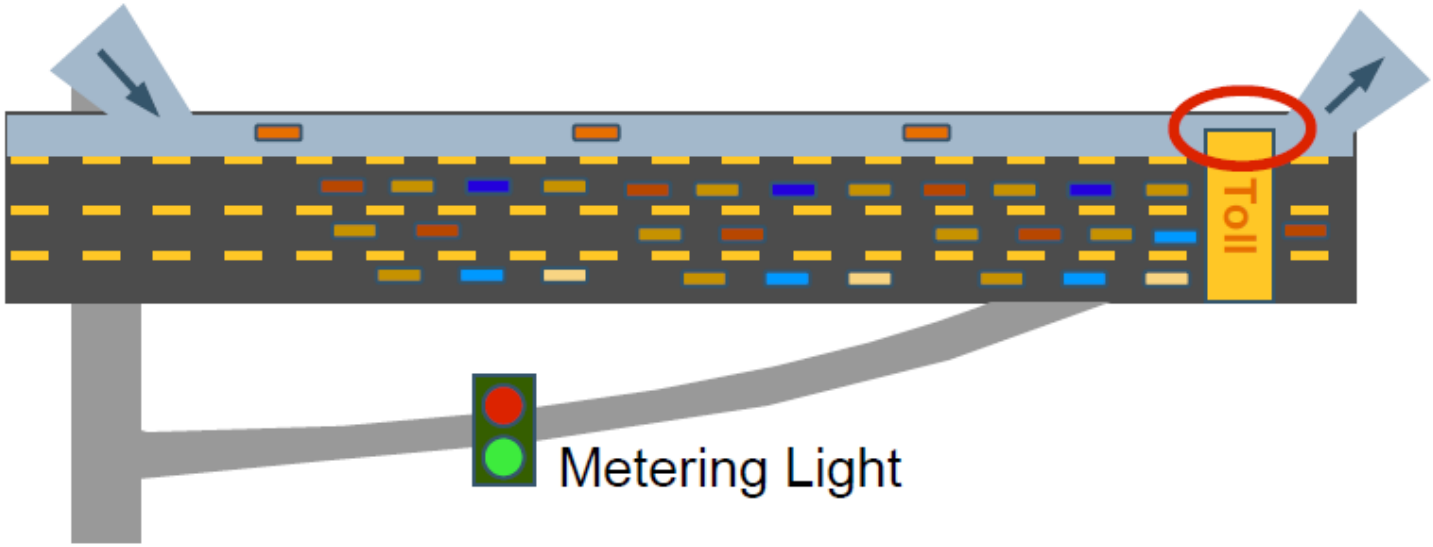


コミューターハイウェイでは、通行の流れやスループットをコントロールするために制御信号を利用している

多くの商用システムにおいても、スループットを確保するために「通行量」を制御してサーバの負荷を最小化する、といった同様のアプローチをしている

アナロジー：リアルタイムスレッド

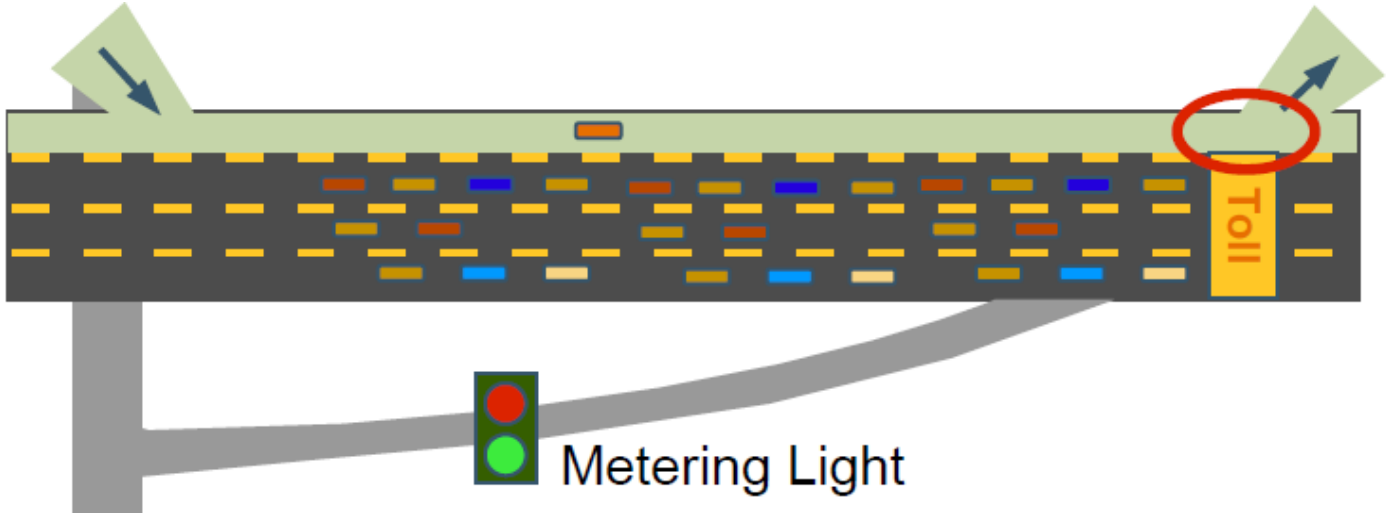
リアルタイムスレッドは通勤レーンのようなもの（合流なしで出入りできると仮定）――所要時間は改善される



しかし、このレーンは他の車（スレッド）と共有される場合があり、通行量が増えればレーンが埋まって速度が落ちる可能性がある

アナロジー：非ヒープリアルタイムスレッド

非ヒープリアルタイムスレッド (NHRT) は個人専用レーンのようなもの――割り込みがないため、正確に所要時間を予測できる



RTSJ では NHRT 、リアルタイムスレッド、通常のスレッドを 1 つのシステム上で扱うことが可能である

リアルタイムスレッドの生成の例

```
import javax.realtime.*;

public class RTThreadSample {
  public static void main(String[] args) {
    // スレッドの優先度を最大に設定
    SchedulingParameters scheduling =
      new PriorityParameters(PriorityScheduler.MAX_PRIORITY);
    // スレッドのデッドライン・コストは無し
    ReleaseParameters release = new AperiodicParameters(null, null, null, null);
    // スレッドに対するメモリ割り当て制限は無し
    MemoryParameters memory =
      new MemoryParameters(MemoryParameters.NO_MAX, MemoryParameters.NO_MAX);
    // スコープメモリエリアを使用
    MemoryArea area = new LTMemory(10000, 10000);
    // プロセッシンググループは指定しない
    ProcessingGroupParameters group = null;
    // Runnable インターフェースを持つオブジェクトを指定
    Runnable logic = new MyThread();

    RealtimeThread rt =
      new RealtimeThread(scheduling, release, memory, area, group, logic);
    rt.start();
    try {
      rt.join();
    } catch (Exception e) {}
  }
}
```

メモリ管理モデルの拡張

- 各メモリモデルでオブジェクトのライフサイクルが異なっている
- RTSJ で定義されているメモリモデル
  - Heap Memory
    - GC の対象
  - Scoped Memory
    - GC の対象外だが、スコープを抜ければ回収される
  - Immortal Memory
    - GC の対象外、決して回収されない

スコープメモリ

- スコープを指定するための複数の方法
  - ScopedMemory オブジェクトの enter() にて指定
  - RealtimeThread 作成時に ScopedMemory オブジェクトを指定→スレッドの生存期間がスコープ
- スコープメモリの破棄のタイミング
  - スコープへの参照カウントが 0 になった時
  - 参照カウントは、ScopedMemory オブジェクトの enter() が呼ばれるたびに 1 つ増え、enter() から抜けると 1 つ減る

スコープメモリの利用例

```
import javax.realtime.*;

// LTMemory はオブジェクトの大きさに対してリニアな時間で
//  割り当てを行うアルゴリズムを持つスコープメモリです。
ScopedMemory sm = new LTMemory(10000, 10000);
sm.enter(new Runnable() {
  public void run() {
    //  このメソッド内のメモリ割り当ては、すべて
    // LTMemory で確保されたメモリエリアから行われます。
    String str = new String("abc");
    ...
  }
}
```

メモリアクセスのルール

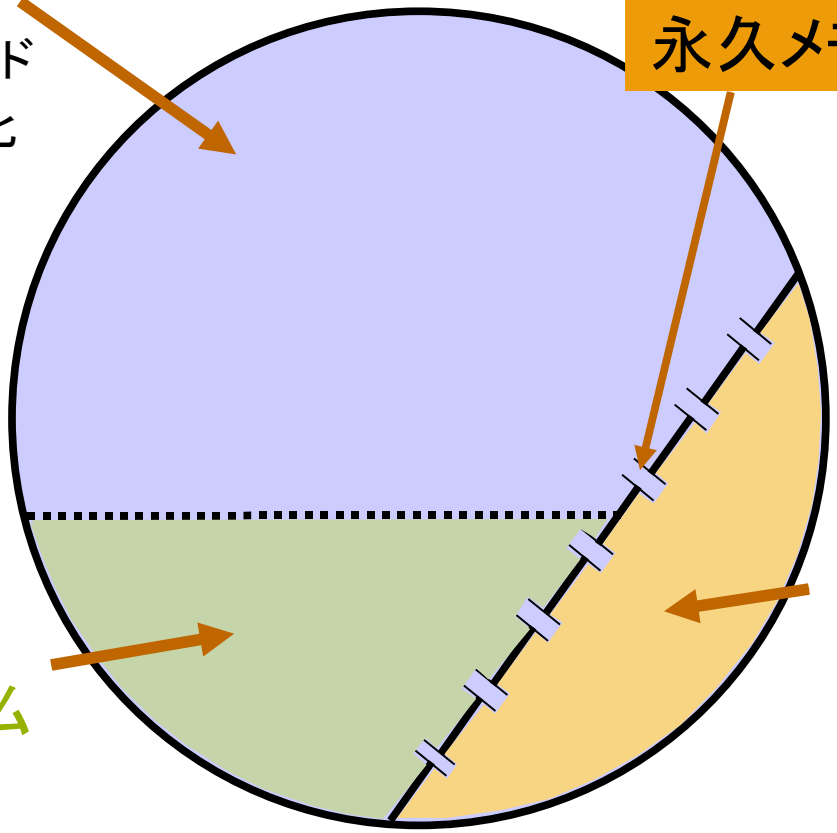
- Scoped Memory に確保されたオブジェクトの生存期間は特殊なため、下のようなルールが適用される

保存場所	Heap の参照	Immortal の参照	Scoped の参照
Heap	Yes	Yes	No
Immortal	Yes	Yes	No
Scoped	Yes	Yes	同じまたは外側のスコープのみ
ローカル変数	Yes	Yes	Yes

RTSJ システムモデル

Java ヒープ
非リアルタイム
- 通常の Java スレッド
- スループット最大化

データ転送キュー
永久メモリ



スコープメモリ
ソフトリアルタイム
- リアルタイムスレッド
- リアルタイム GC

スコープメモリ
ハードリアルタイム
- 非ヒープリアルタイム スレッド (NHRT)
- 遅延の上限設定

無待機キュー

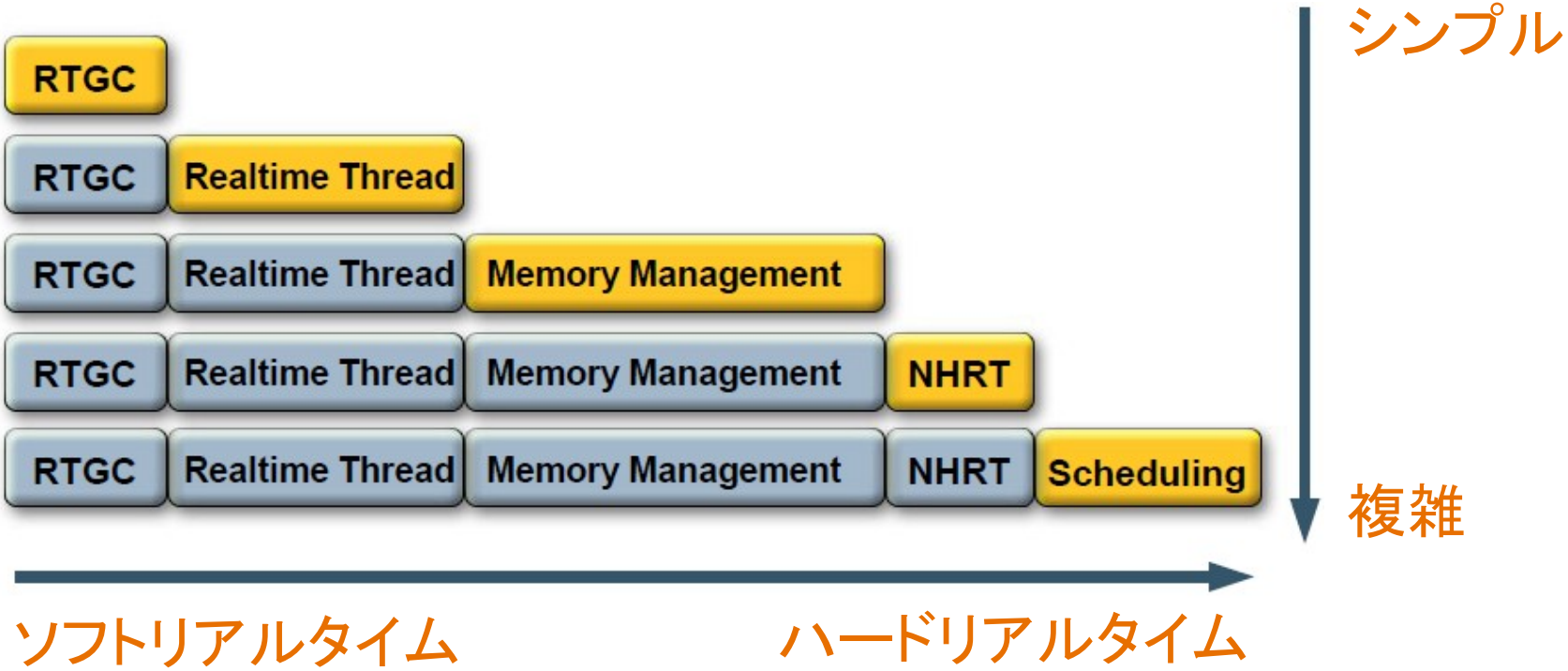
```
package javax.realtime;

// write をブロックしない無待機キューです
//  ヒープを使わないスレッドからヒープを使うスレッドにデータを渡すときに
//  使います
public class WaitFreeWriteQueue {
  public WaitFreeWriteQueue(java.lang.Runnable writer,
                            java.lang.Runnable reader,
                            int maximum, MemoryArea memory);
  ...
  //  キューの先端から要素を取り出して返します
  //  キューが空のときは要素が取り出せるまでブロックします
  public java.lang.Object read();
  //  キューに要素を入れて  true  を返します
  //  このメソッドはブロックしません。キューがフルのときは何もせず
  // false  を返します
  public boolean write(java.lang.Object object);
  ...
}
```

プログラムに依存しないアプローチ

- VM 実装の改良によるリアルタイム性の向上
  - リアルタイム GC (RT GC)
    - GC の細分化・並列動作
    - 一定時間内の GC 動作の制限
    - 動的な GC スレッド優先度の変更
  - クラスの事前ローディング
    - あらかじめクラスを指定しておき、VM 起動時にローディングを行う
  - クラスメソッドの事前コンパイル
    - あらかじめメソッドを指定しておき、VM 起動時 (AOT) またはクラス初期化時 (ITC) にコンパイルしておく

リアルタイム Java 利用オプション

- 必ずしもすべての機能を使う必要はない
  - > リアルタイム性能とシンプルさのトレードオフ

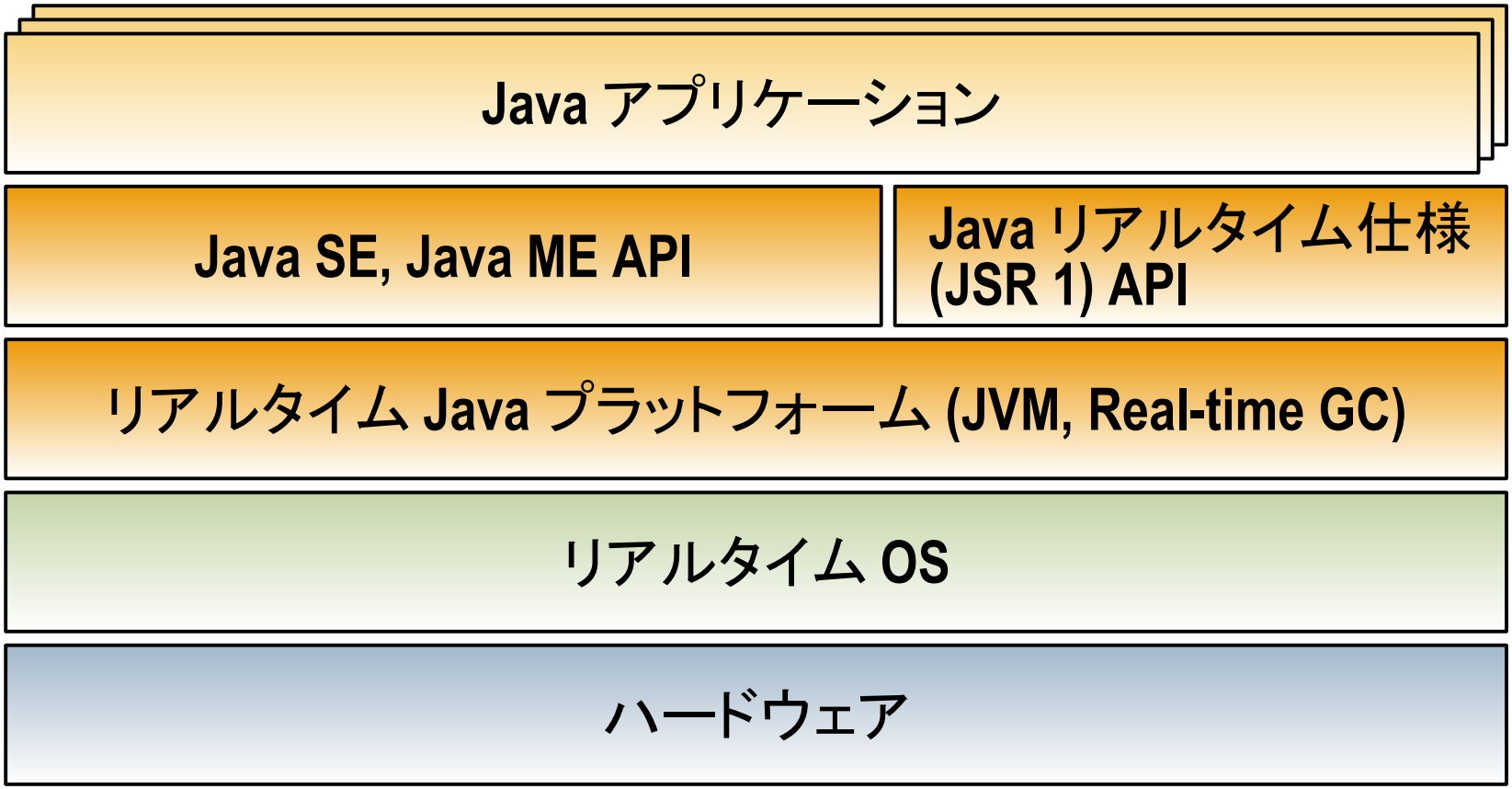


ターゲット分野

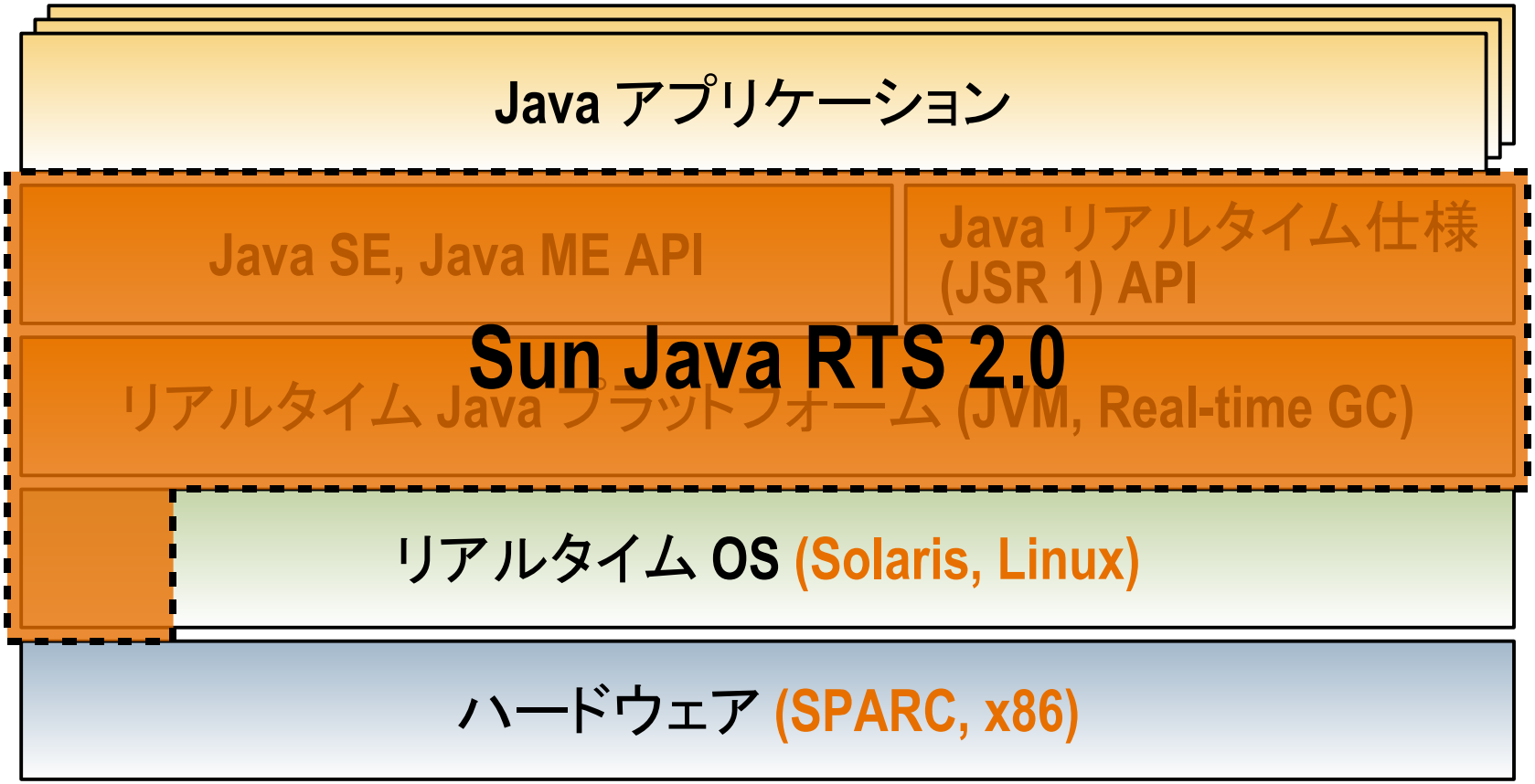
- 軍事システム
  - GC の最中にミサイルが飛んできたら…
- 金融システム
  - 金融取引における QoS および規制への対応
- 通信システム
  - VoIP・PBX・IMS・新しい 3G サービス
- 産業システム
  - 産業オートメーション・プラント・プロセス管理

Sun Java Real-Time System 2.0

リアルタイム Java 環境の構成



Sun Java Real-Time System 2.0



Sun Java Real-Time System 2.0

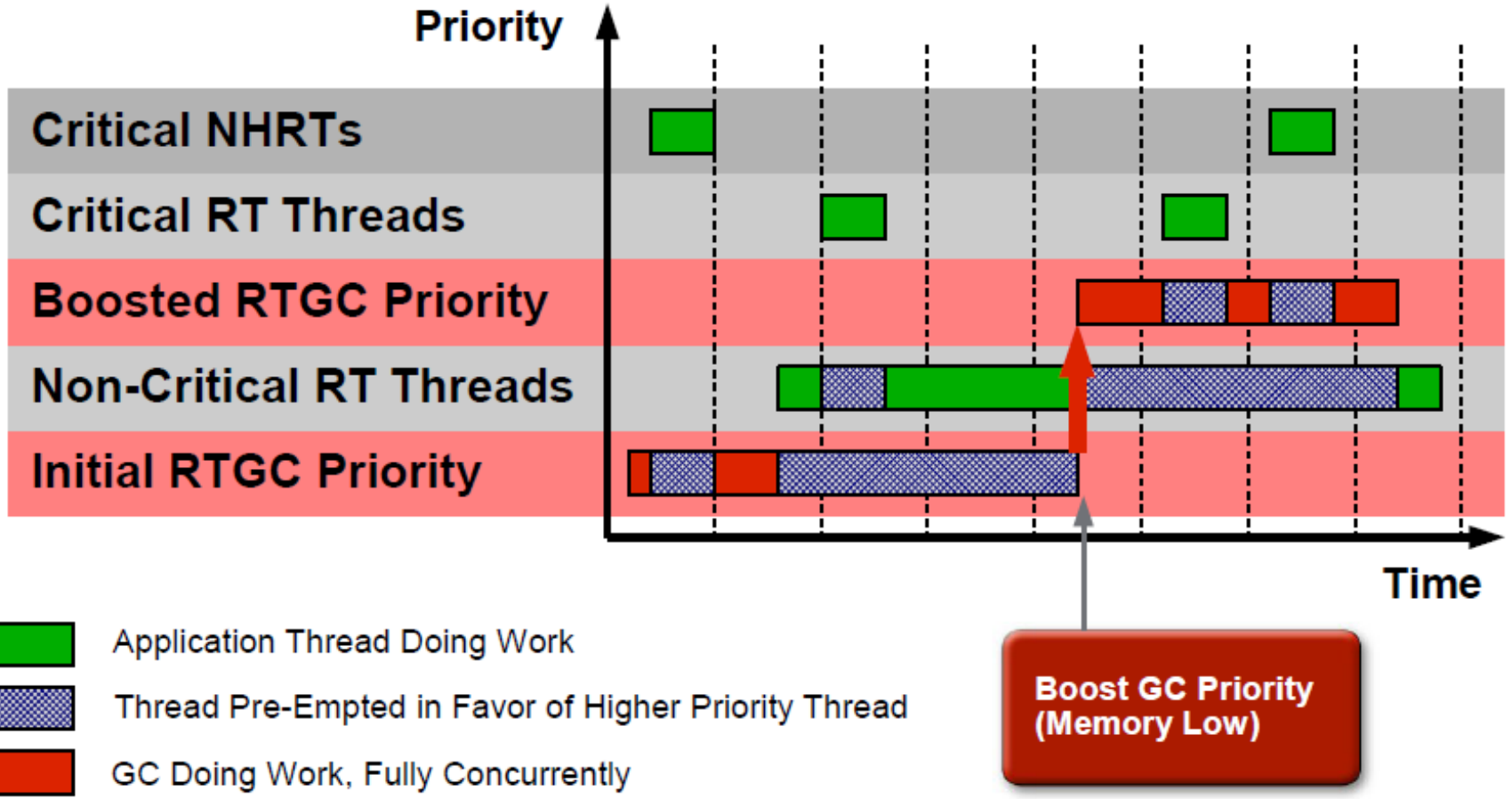
- Java Platform, Standard Edition (Java SE プラットフォーム) 5.0
- Solaris™ オペレーティングシステム (Solaris OS) 10 および Linux（予定）
- x86, x64 および SPARC® アーキテクチャ
- リアルタイムガベージコレクタ
- NetBeans モジュール

Sun Java RTS —— RTGC

ガベージコレクションにポリシーを適用

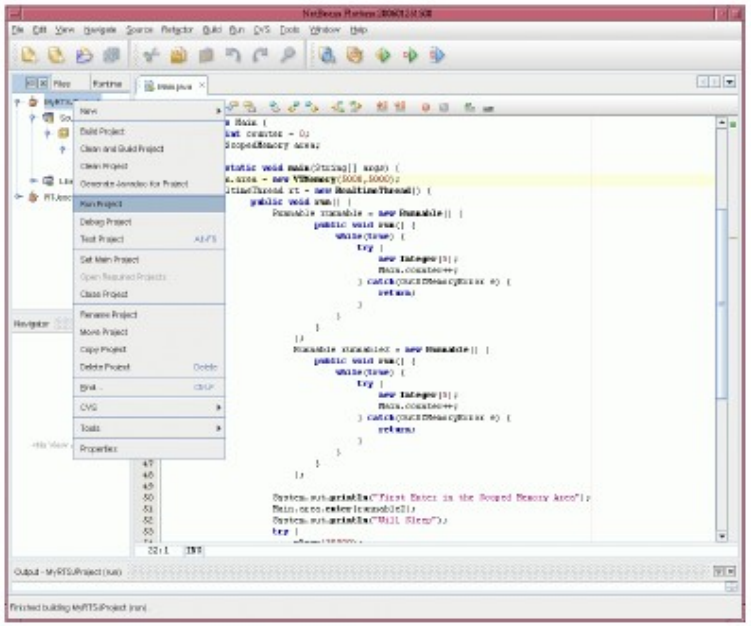
- ゴール：優先度の高い GC 対象スレッドの遅延を最小化
  - GC スレッドの優先度を柔軟に変える
  - ヒープの優先的な配分
- 特長
  - スケーラブル：マルチプロセッサ環境でも問題なし
  - 柔軟性：優先度の低い RT スレッドに対してはいろいろなポリシーを適用できる
    - メモリ消費が大きくなった場合、GC のオーバーヘッドはこれらのスレッドが負担する

RTGC スケジューリング (1CPU)

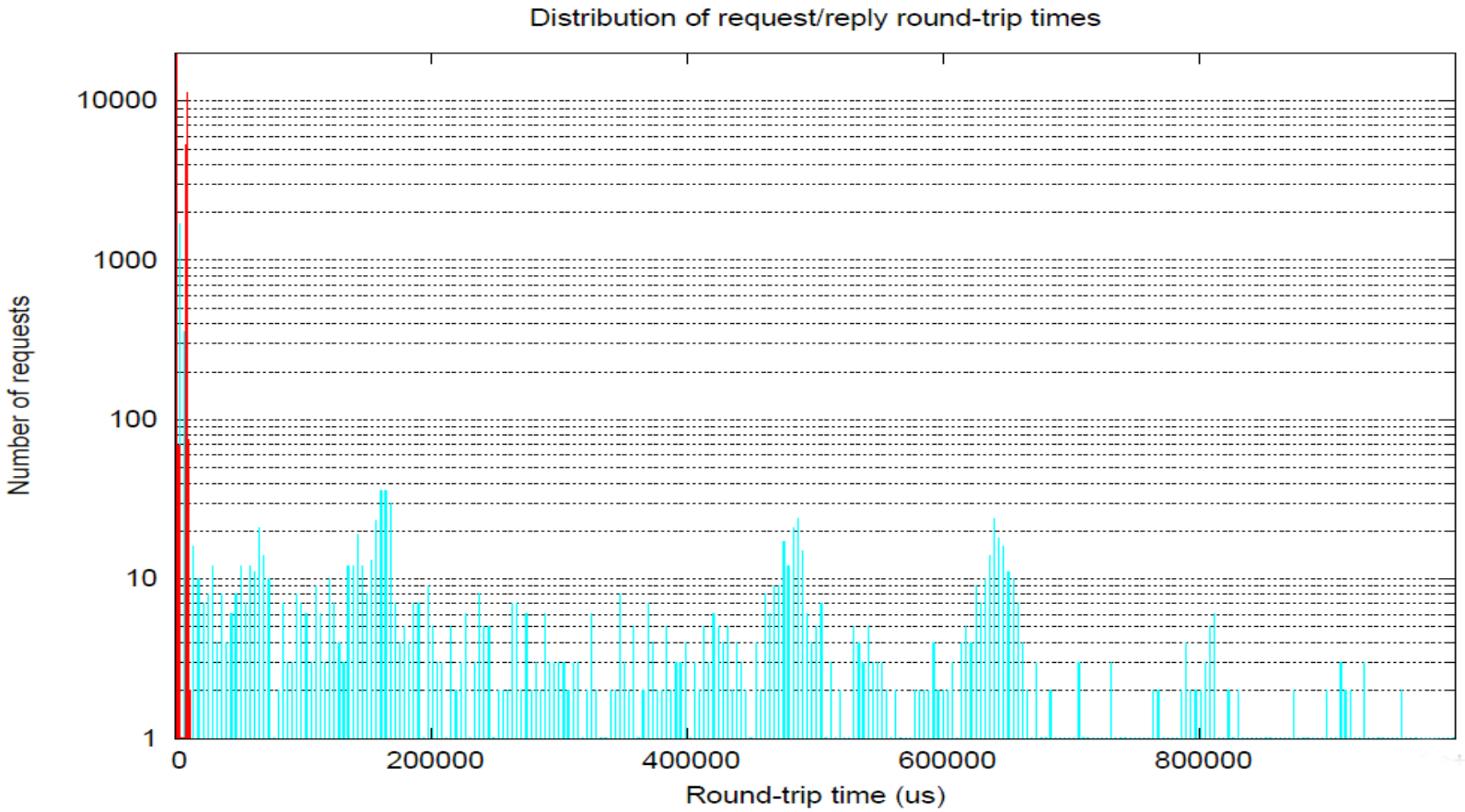


ソフトリアルタイムのプログラミング

- Java RTS プラグインをダウンロード、で
  - > ホスト上でクロス開発
  - > ネットワーク経由でデプロイ
  - > ターゲット上で実行
- … を NetBeans IDE で できます



Real-Time Application Server
標準 Java SE と Java RTS の比較



Java RTS – すべてのトランザクションは 11 ミリ秒以内に完了

標準 Java SE – トランザクションは 11 ミリ秒から 3.5 秒の間に分布

まとめ

- リアルタイムシステムとは指定された時間内に処理を完了できるシステムである
- リアルタイム Java 仕様では新しいメモリモデル、スレッドスケジューリングなどの導入により、実行の予測可能性を格段に向上させる
- リアルタイム GC・事前コンパイル等の VM の改良によりさらなる性能の向上が得られる
- Sun Java Real-Time System をお試しください
  - 評価版ダウンロード
    http://java.sun.com/javase/technologies/realtime/rts/

Real-Time Java
～組み込みから エンタープライズまで～

サン・マイクロシステムズ株式会社
草薙 昭彦
Akihiko.Kusanagi@Sun.COM