Java GUI Application using @PT – Group 27

Presented by:
Aneesh Thunga
Chris (Hanliang) Ding
Jerry Fan

What is @PT…

…and what is it used for?

Presented by Chris

Parallel Programming

- Language Constructs

  -Intuitive for programmers

  -Consistent programming and coding styles

  -Less control over parallelization of the code

- Library API

  -Complex for the programmers to piece together all the parallel blocks

  -Responsibility falls on programmer and inconsistent coding styles

  -More control over parallelization

What is @PT

- Annotation based
- Extension of ParaTask

- **Intuitive for programmers** - Combine tasking constructs from ParaTask into annotation constructs
- **Control over parallelization** - Many parallelization techniques incorporated into the annotations
- **Flexibility** - Annotation at invocation of method instead of declaration

```
@Future
Void result = workerFunction(int input);
```

@Future

- Objects whose value will be available in the future
- One-off tasks

```
@Future
Void result = workerFunction(int input)
```

- I/O tasks

```
@Future(taskType = TaskTypeInfo.INTERACTIVE)
Void result = IOWorkerFunction(int input)
```

- MultiTask - can be MULTI_IO or MULTI

```
@Future(taskType = TaskTypeInfo.MULTI_IO, taskCount = 2)
Void result = MultiWorkerFunction(int[] input)
```

@Future groups

- Synchronization points

-When accessing return value the task has to finish execution

Future group

Each task added is future

```
public Boolean runAsync(Void wait) {
    @Future(taskType = TaskInfoType.INTERACTIVE)
    Boolean[] taskGroup = new Boolean[detections.length];

    for (int i = 0; i < imageBytes.length; i++) {
        @Future(taskType = TaskInfoType.INTERACTIVE)
        Boolean task = asyncWorker(i);
        taskGroup[i] = task;
    }

    return taskGroup == null;
}
```

-Accessing elements of Future groups return values from require the single element to be completed

- Accessing the whole array every object in the group to be completed

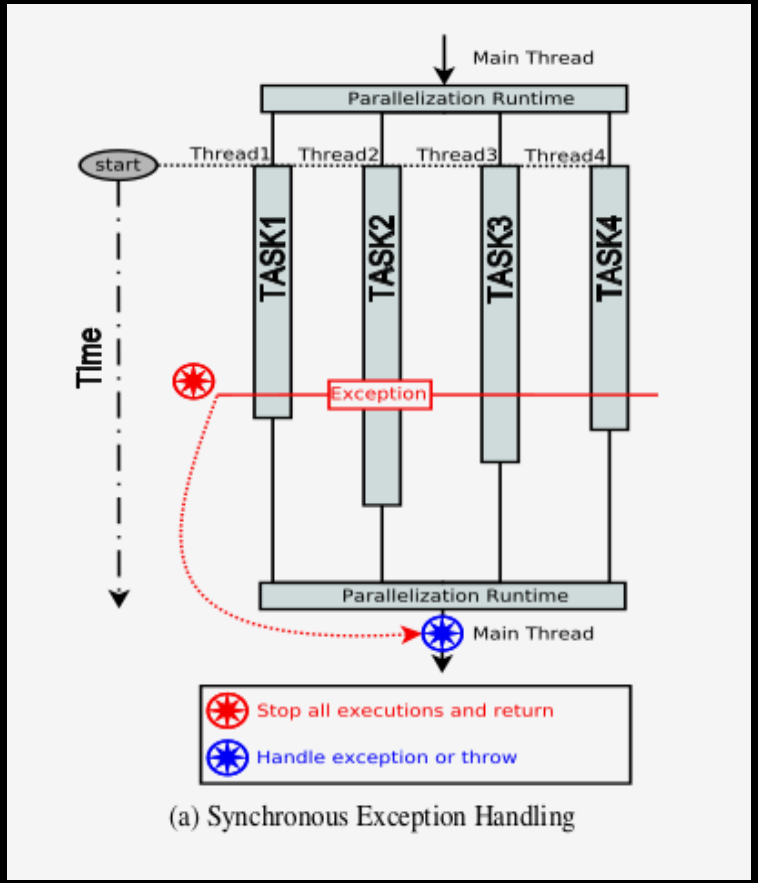
Reduction

- Implemented with RedLib - lightweight ready to use reduction library
- Supports many different reduction operations
- Supports many multi-layer and complex data structures

```
@Future(taskType = TaskInfoType.MULTI, reduction = "union(union(max))")
Map<K, Map<T, Integer>> wordCount = countWords(documents);
```

Error Handling

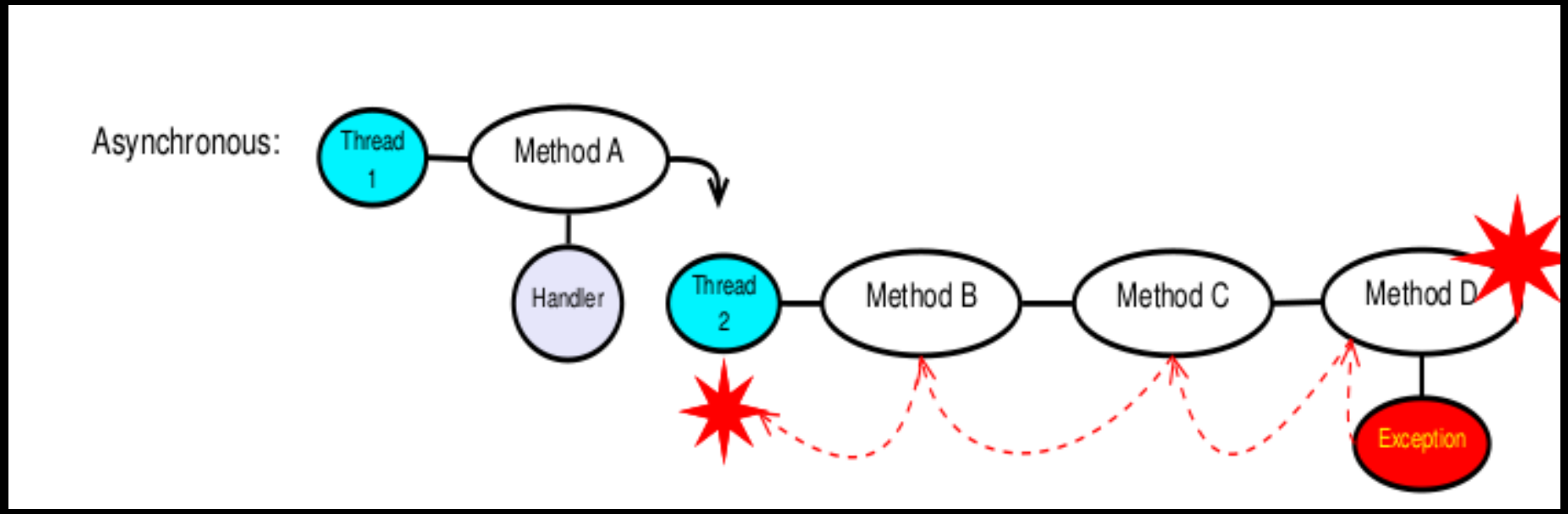
- Synchronous approach stops all tasks from executing



(a) Synchronous Exception Handling

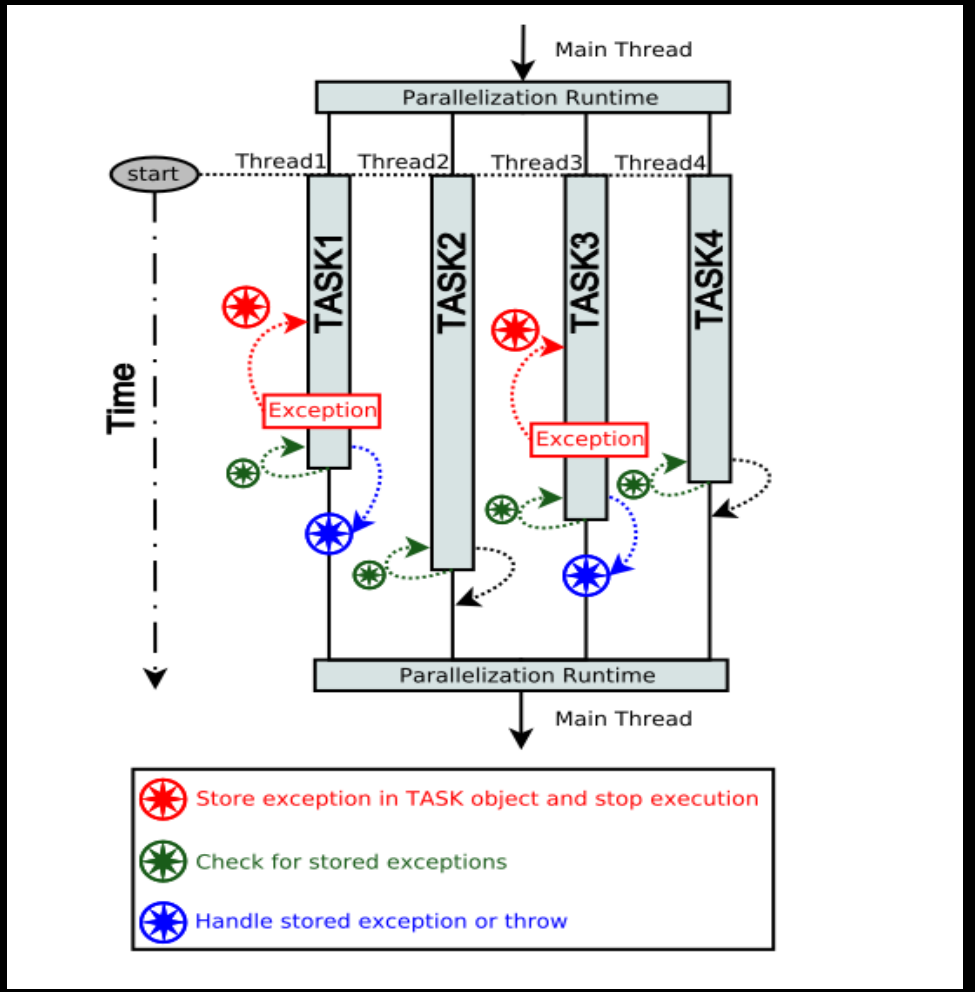
Asynchronous Error Handling

- Different threads handling tasks can disjoint the call stack



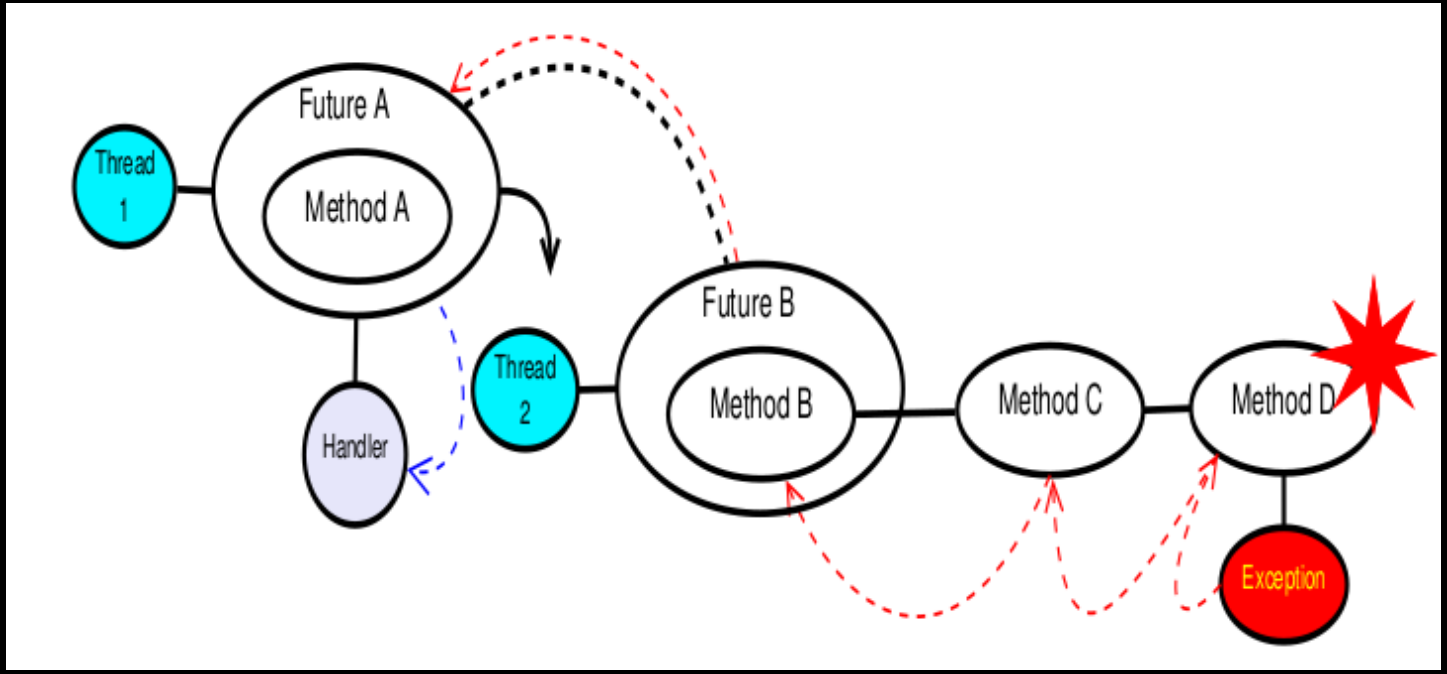
@AsyncCatch

- Stores the exception in the task object
- Handles exceptions after task completion independent of other threads



@AsyncCatch

- Future objects create pointer to calling future object which solves disjointed call stacks



@AsyncCatch

@AsyncCatch annotates a future object for asynchronous exception handling

```
@AsyncCatch(throwables={RuntimeException.class},handlers={"handleRuntimeEx()")
@Future
Void result = worker(int input);


private void handleRuntimeEx() {
    //Exception handling code
}
```

GUI Operations

Three main types of GUI Operations:

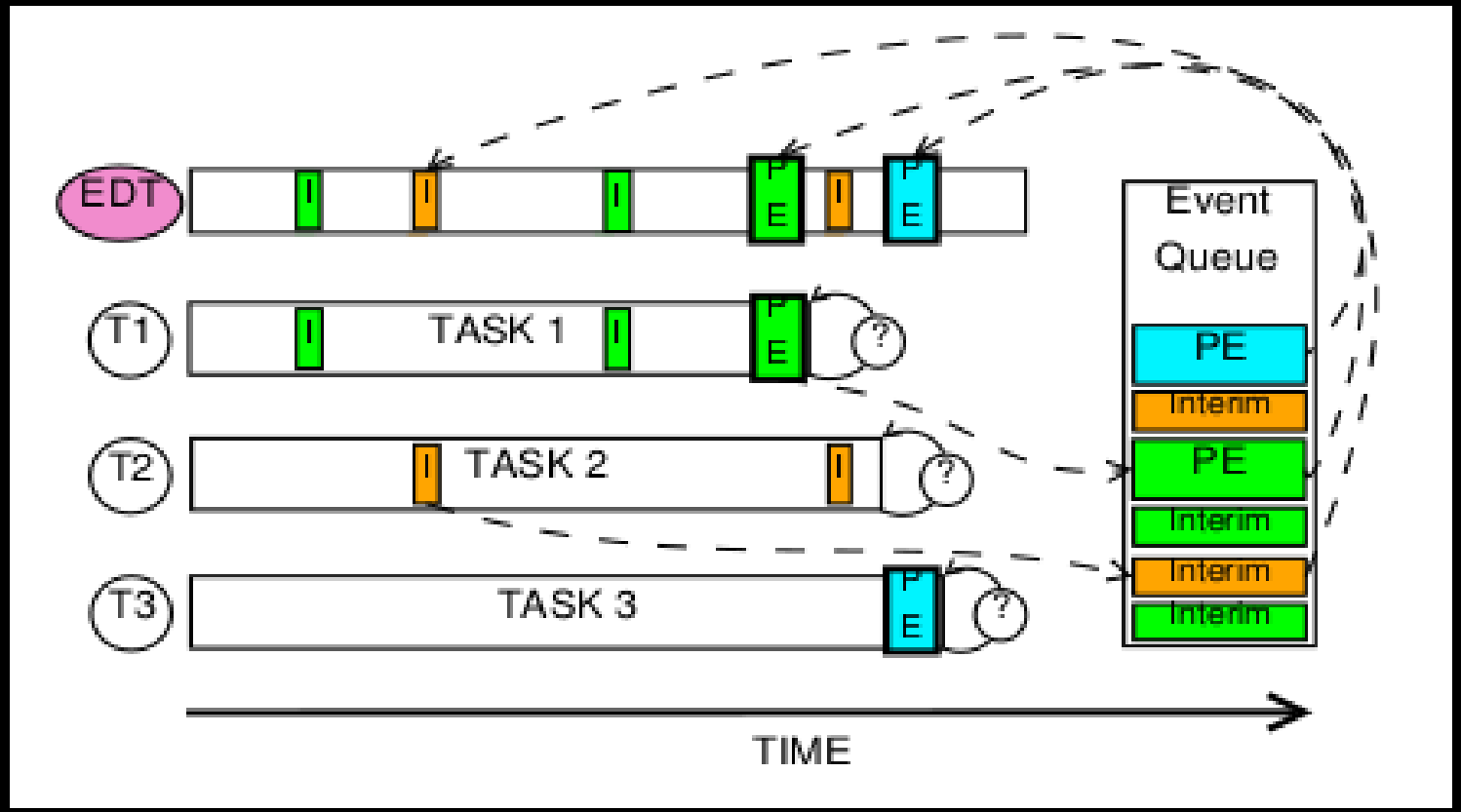
- **Post Execution** – results after a task is complete
- **Interim** – results as the task is running. e.g. progress bar
- **Finalization** – After termination of an entire program

@Gui

- Combines all three types of operations into one constructs
- Can take in any number of arguments
- Included in the sequential code
- Programmers don't need to worry about syntactic differences between EDT and Computational threads

@Gui

Solves timing issues by only Immediately enqueueing Interim type operations



@Gui

```
@Future
int task1 = worker(int input);

@Gui
Void change = updateGui(task1)
```

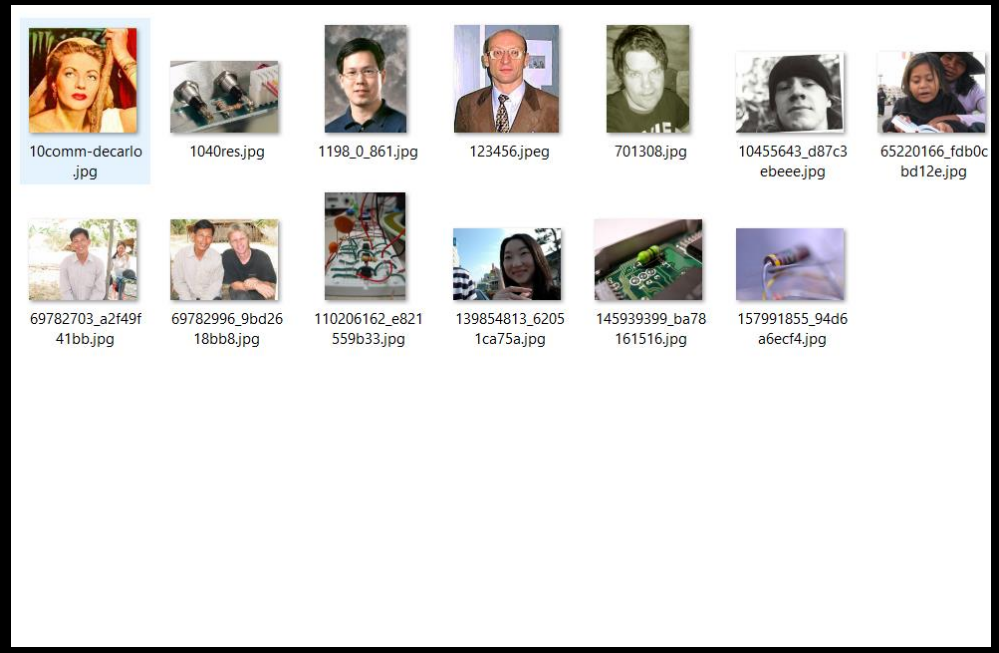
GUI Application

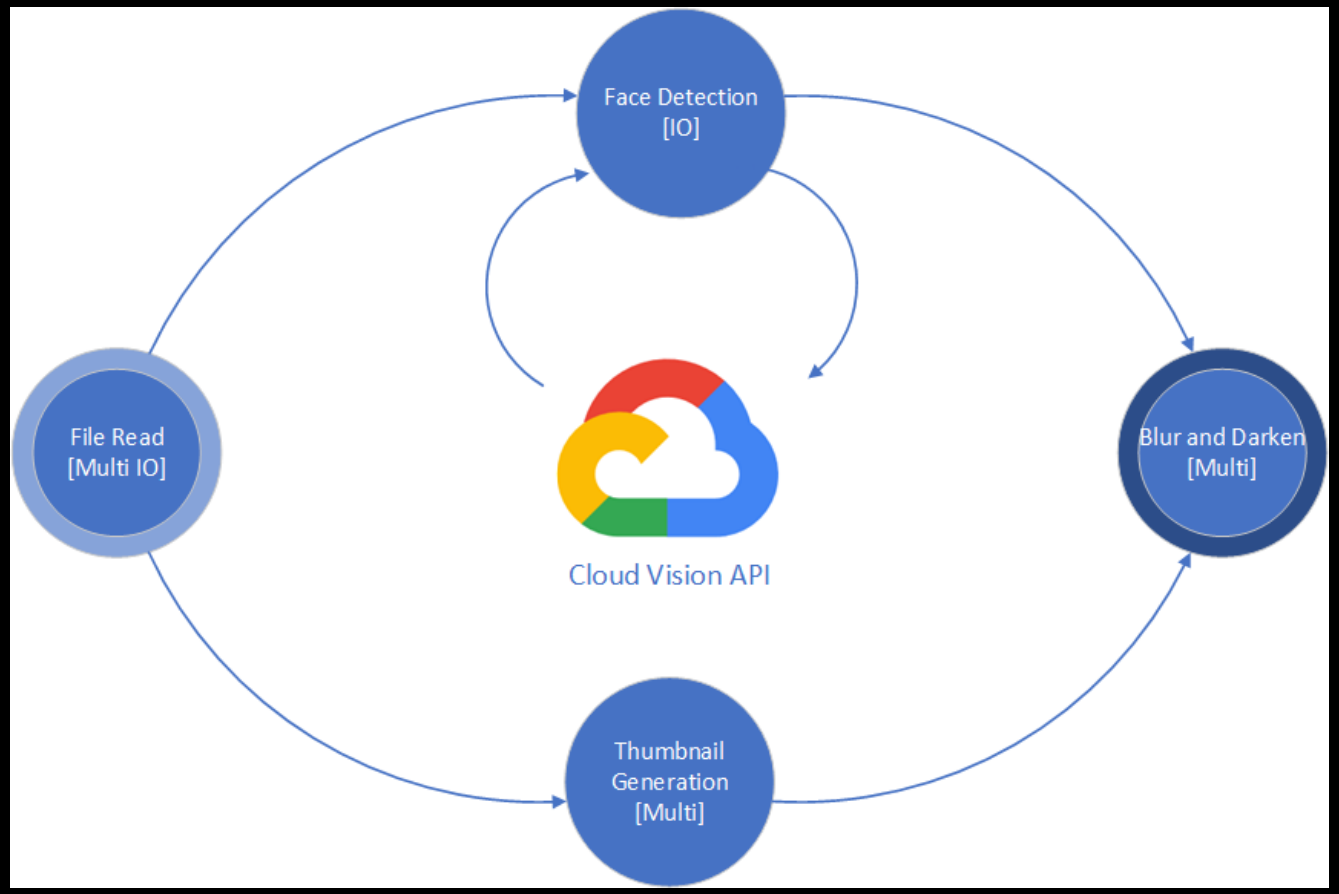
Face Gallery – An photo gallery that recognizes faces

Presented by Jerry

Purpose

- Sorting through photos in a folder visually
- Identifying the faces from the collection
- Test disk read operation in different run modes
- Test network requests in different run modes (Google Cloud API)
- Test CPU intensive operation in different run modes



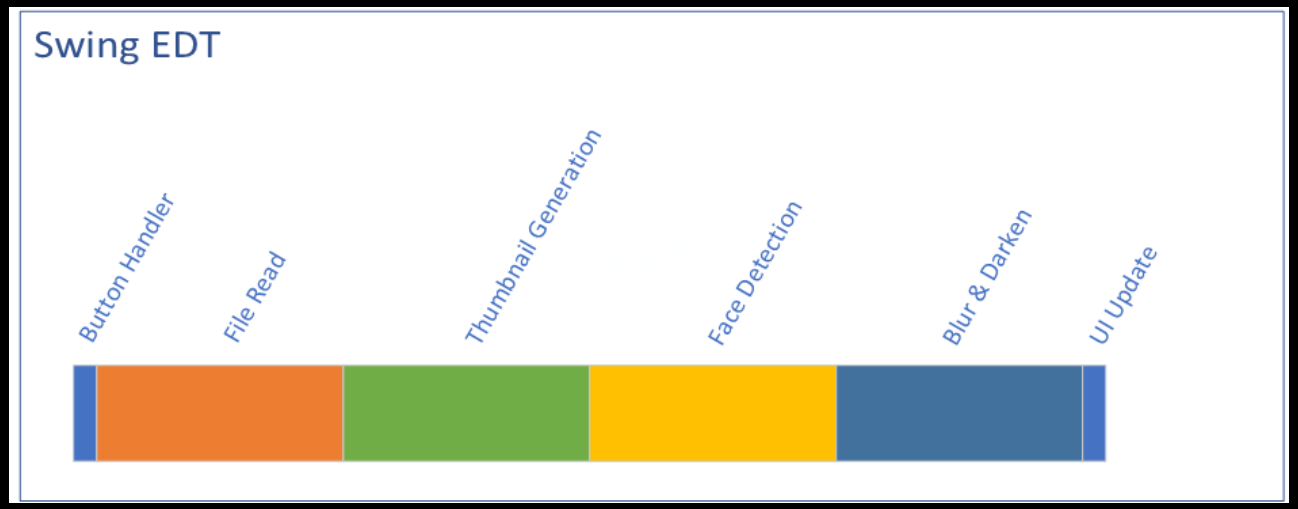


Task Decomposition

- File read
- Thumbnail generation
- Face detection
- Dim and blur

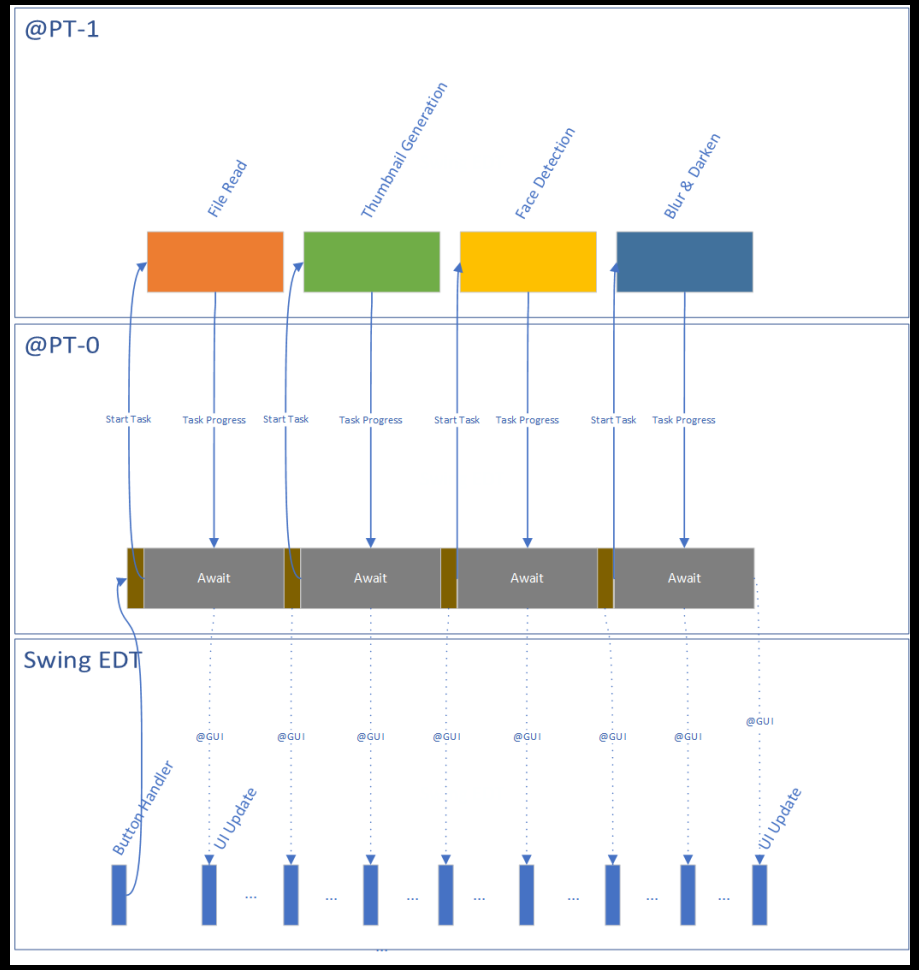
Run Modes

- Sequential
  - Blocking
- Concurrent
  - Non blocking
  - Two threads
- Parallel
  - Non blocking
  - Multiple threads
- Parallel Pipeline (NOT implemented)
  - Non blocking
  - Multiple threads
  - No synchronisation between task groups



Sequential

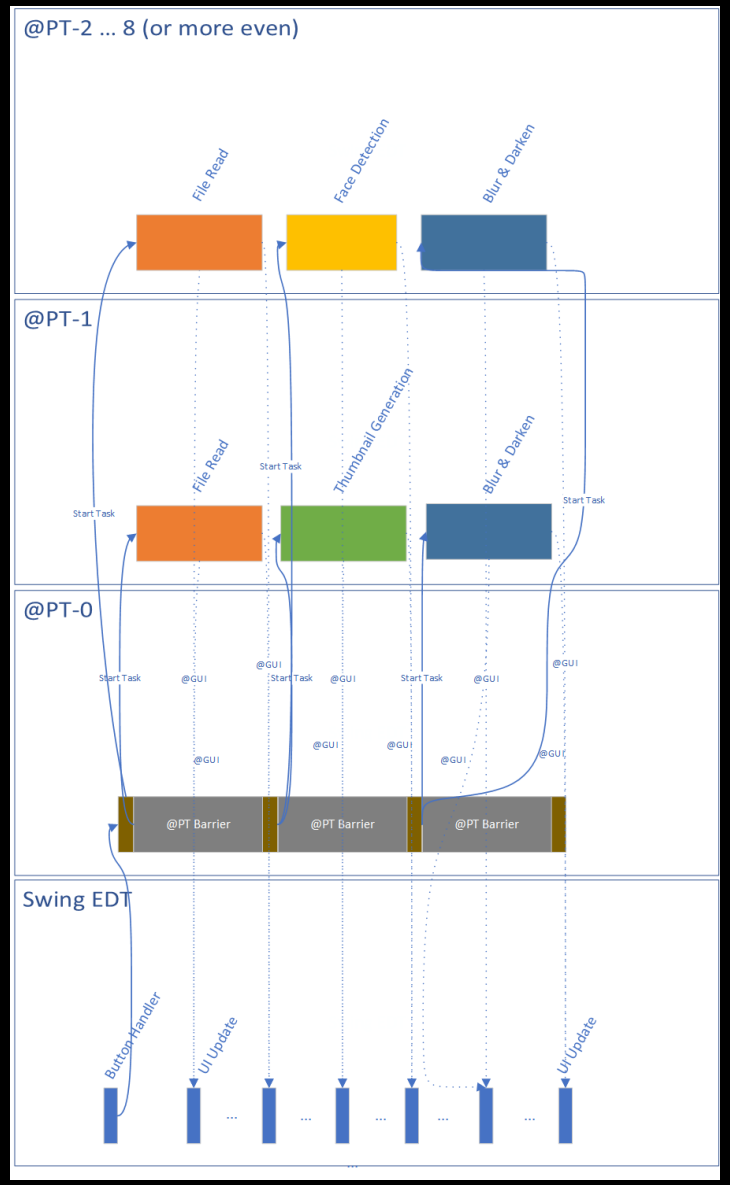
- No use of @PT
- Runs on the Swing ED thread
- Application is blocked
- Non responsive GUI
- Display is updated when all processing finishes



Concurrent

- @PT one-off task
  - Actually uses between 1 – 2 of these
- Await with BlockingQueue
  - Not using @PT barrier
- Intermittent GUI updates
- Individual task progress displayed

```
@Task
public Void performConcurrent(Function<TaskerStats, Void> statsUpdater, Function<List<BufferedImage>, Void> imagesUpdater, boolean batchFaceDetect)
```



Parallel

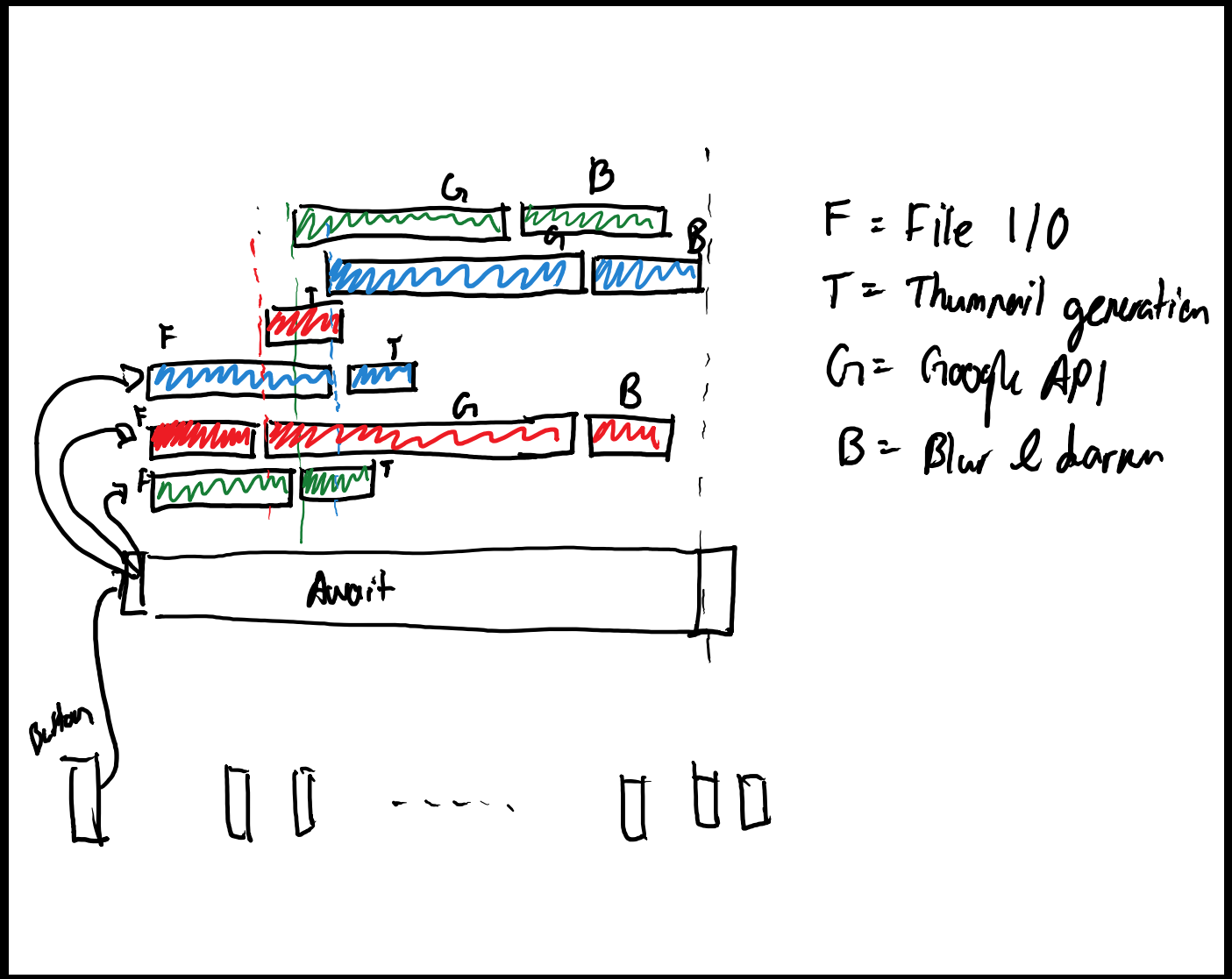
- @PT one-off task
  - Used to manage GUI updates
- Await with @PT's barrier
  - Still using blocking queue within the individual tasks
- Intermittent GUI updates
- Individual task progress displayed

```
@Future
Void t0 = fileRead(statsUpdater);

@Future(depends = "t0")
Void t1 = thumbnailGenerate(statsUpdater, imagesUpdater);

@Future(depends = "t0")
Void t2 = faceDetect(statsUpdater);

@Future(depends = "t1,t2")
Void t3 = rescale(statsUpdater, imagesUpdater);
```



Parallel Pipeline

- Hard to draw a diagram of
- NOT currently implemented
- Idea to speed up processing
- No task await
  - Each work item awaits individually
- Intermittent GUI updates
- Individual task progress displayed
- Any task may start at any time for a given work item (image)

Batch API Call

- Supported by Google Cloud Vision API (Max 16 per batch)
- Process multiple images with one network request
- Can be used in our sequential or concurrent run modes
- Why was this implemented in the application?
  - Unfair to chain network requests
  - More realistic to compare efficient sequential network request against parallel
  - Testing with 1 image per request was taking too long

Application Demo

Presented by Aneesh