Background Tasks

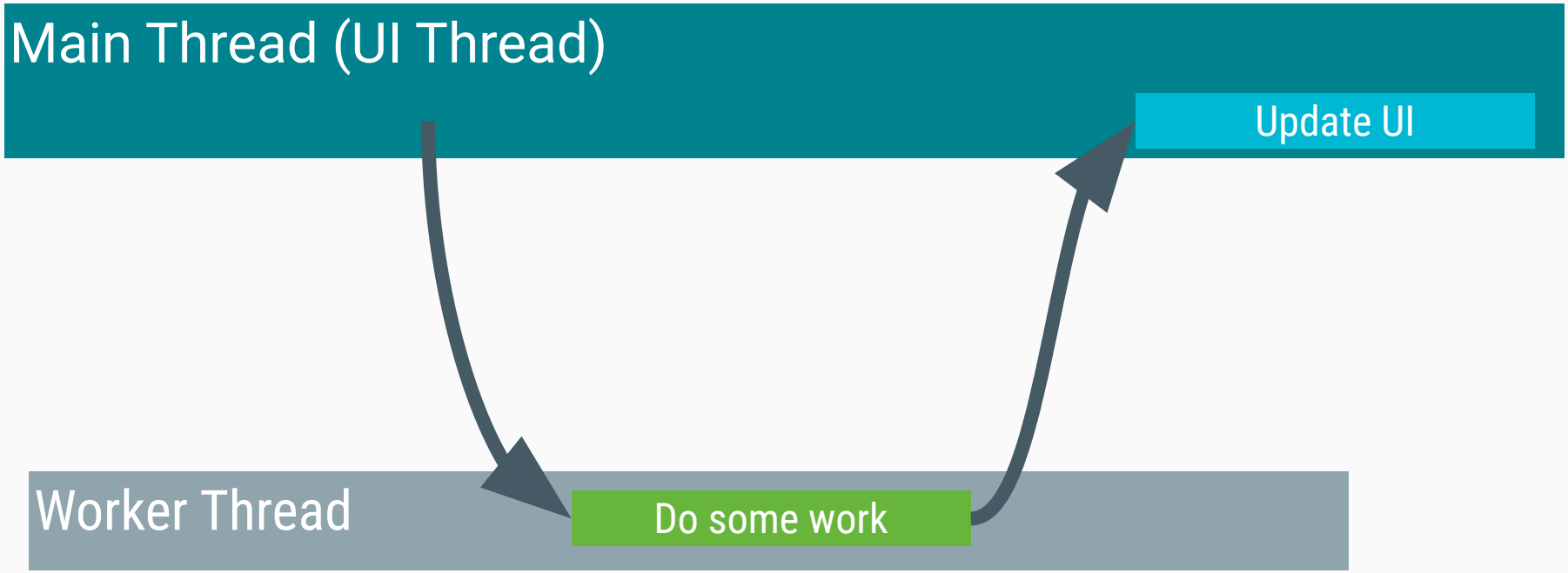
Main thread

- Thread: Chemin d’exécution indépendant exécutant le code ligne par ligne
- Les apps s'exécutent sur un Java thread appelé "main" ou "UI thread"
- Dessine l’UI et réponds aux actions de l’utilisateur
- L’UI thread a 16 ms pour préparer la prochaine update de l’écran sans freeze
- On utilise l’Android UI toolkit seulement sur le main thread

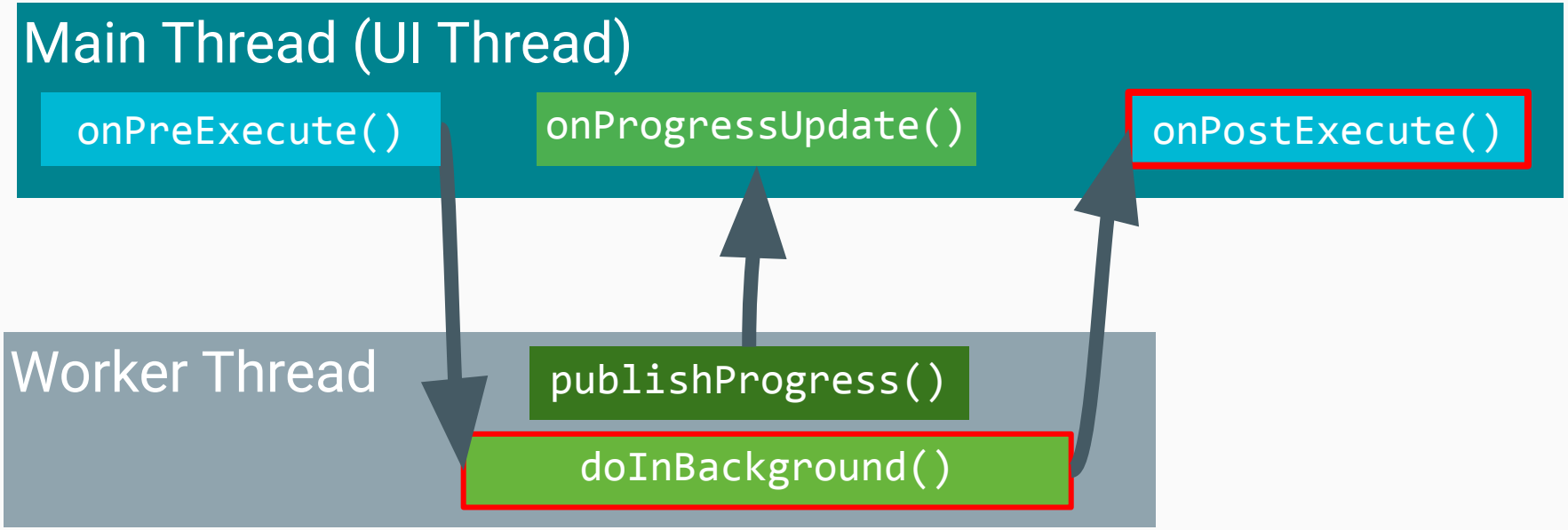
Threading

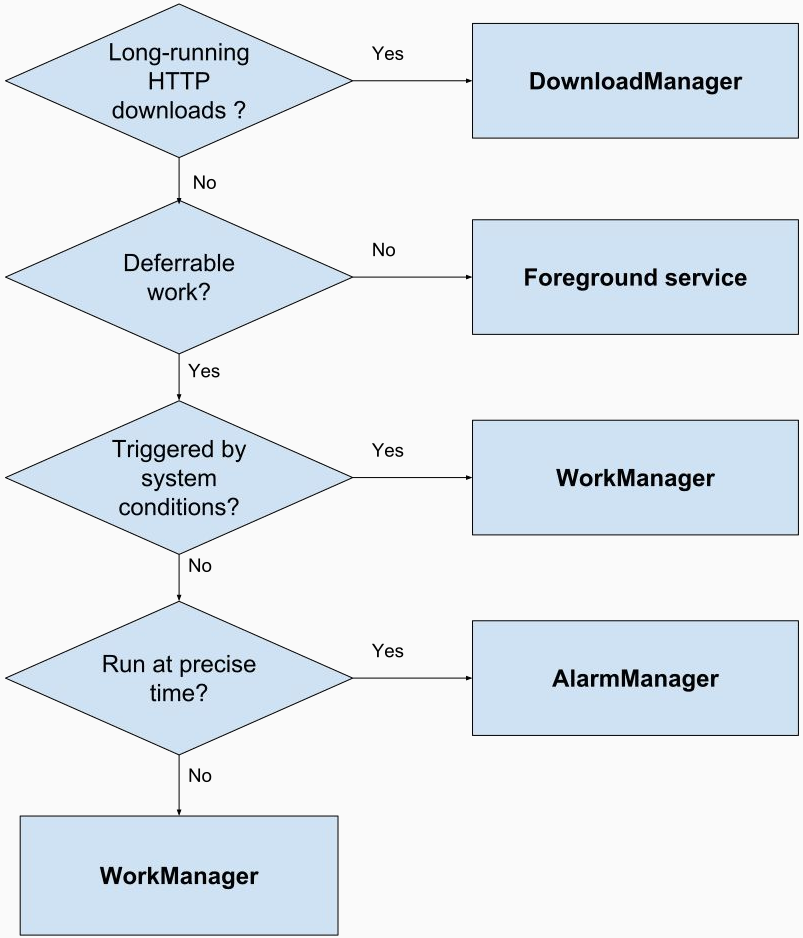
- Thread, Runnable, ThreadPoolExecutor: threads “bas niveau”
- AsyncTask: tâche courte, ou ne renvoyant pas de résultats à l’UI
- Services: Intent, started, bound, background (➡️ JobScheduler), **foreground**
- ~~Loaders, AsyncTaskLoader: tâche longue, renvoyant des résultats à l’UI~~
- Background Service: tâche longue, sans UI
- WorkManager: “nouvelle” façon de gérer les tâches longues et indépendantes du cycle de vie de l’app
- LiveData: Nouvelle façon de récupérer les résultats de tâches parallèles (remplace les usages des Loaders)

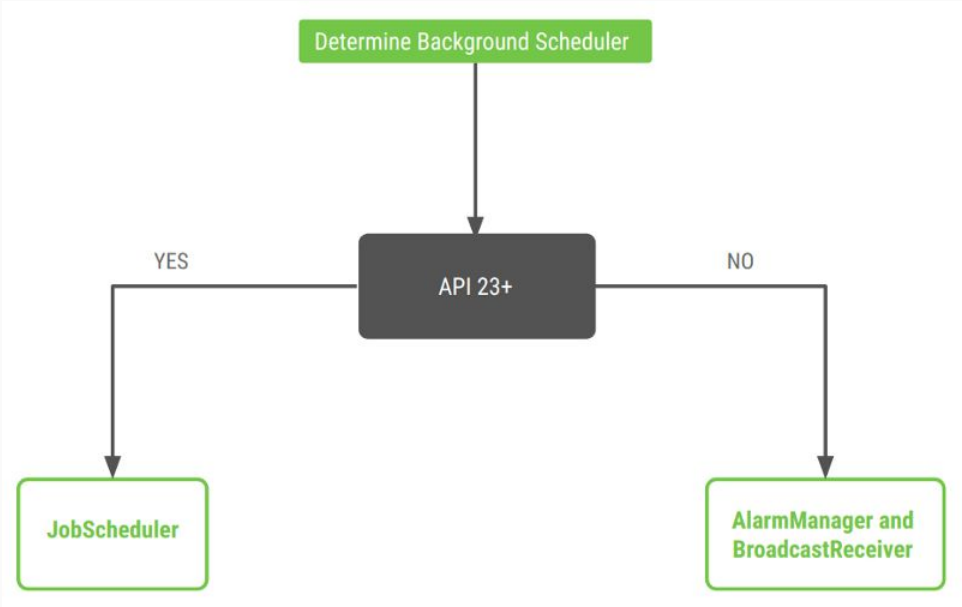
Background threads



Ex: AsyncTask




Long-running HTTP downloads ?
Yes
DownloadManager
No
Deferrable work?
No
Foreground service
Yes
Triggered by system conditions?
Yes
WorkManager
No
Run at precise time?
Yes
AlarmManager
No
WorkManager


Determine Background Scheduler
YES
API 23+
NO
JobScheduler
AlarmManager and BroadcastReceiver

WorkManager

WorkManager: Nouvelle façon de gérer les traitements longs:

- Add work constraints like network availability or charging status
- Schedule asynchronous one-off or periodic tasks
- Monitor and manage scheduled tasks
- Chain tasks together
- Ensures task execution, even if the app or device restarts
- Adheres to power-saving features like Doze mode

```
class UploadWorker(appContext: Context, workerParams: WorkerParameters)
    : Worker(appContext, workerParams) {
    override fun doWork(): Result {
        val imageUriInput = getInputData().getString(Constants.KEY_IMAGE_URI)
        val response = uploadFile(imageUriInput)
        val outputData = workDataOf(Constants.KEY_IMAGE_URL to response.imageUrl)
        return Result.success(outputData)
    }
}

val constraints = Constraints.Builder().setRequiresCharging(true).build()
val imageData = workDataOf(Constants.KEY_IMAGE_URI to imageUriString)
val uploadWorkRequest = OneTimeWorkRequestBuilder<UploadWorker>()
    .setInitialDelay(10, TimeUnit.SECONDS)
    .setInputData(imageData)
    .setConstraints(constraints)
    .addTag("uploads")
    .build()

WorkManager.getInstance(context).enqueue(uploadWorkRequest)
```

```kotlin
class CoroutineDownloadWorker(context: Context, params: WorkerParameters)
    : CoroutineWorker(context, params) {
    override val coroutineContext = Dispatchers.IO // overrides Dispatchers.DEFAULT
    override suspend fun doWork(): Result = coroutineScope {
        val jobs = (0 until 100).map {
            async {
                setProgress(workDataOf("progress" to it))
                downloadSynchronously("https://www.google.com")
            }
        }
        jobs.awaitAll() // awaitAll will throw an exception if a download fails
        Result.success()
    }
}

WorkManager.getInstance(applicationContext).getWorkInfoByIdLiveData(requestId)
     .observe(observer, Observer { workInfo ->
       progressBar.progress = workInfo?.progress.getInt("progress", 0)
})
```