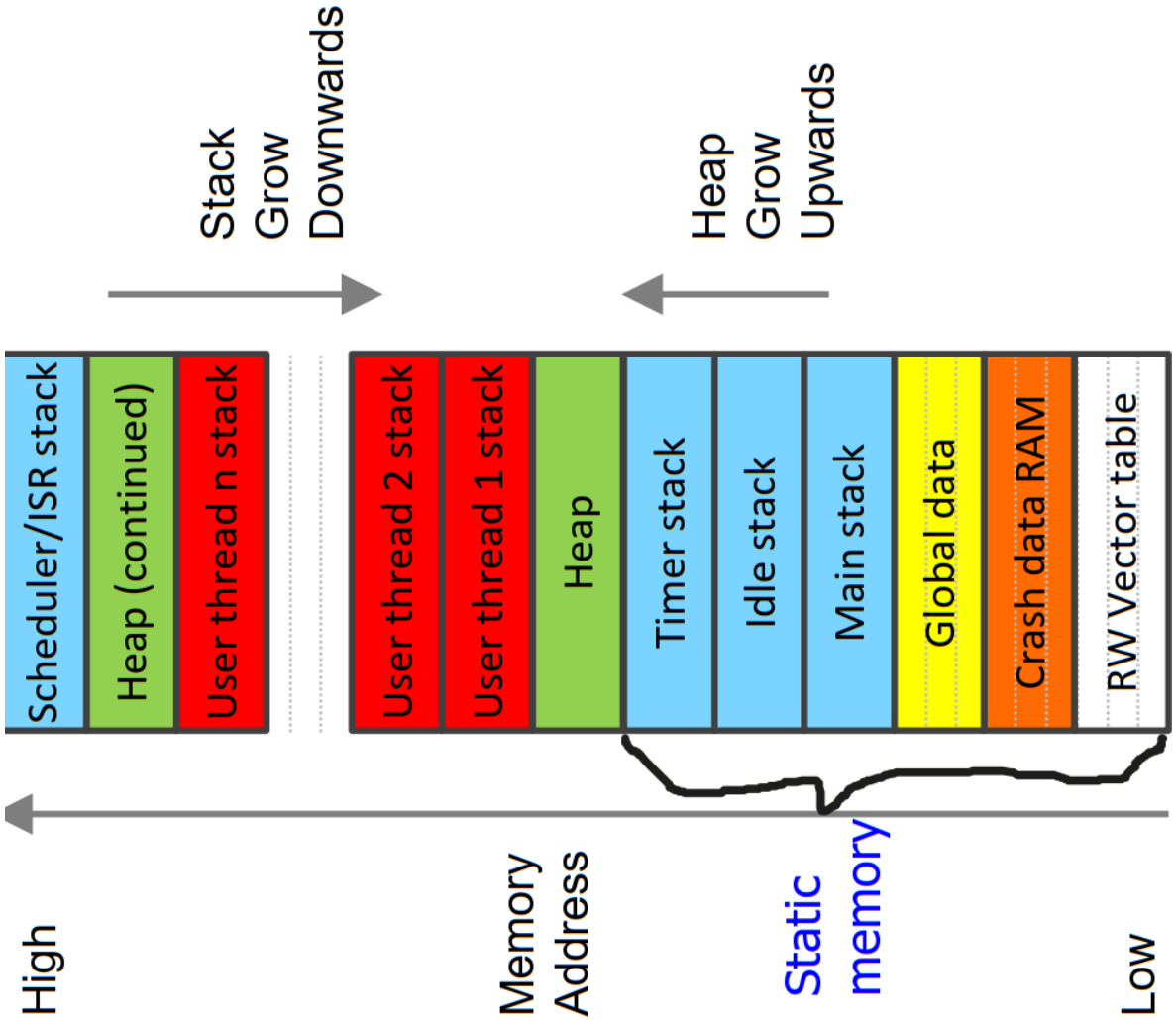
1 Mémoire

- Code ou texte
  - Instructions binaires à être exécutée
  - Elle est habituellement read-only
  - Le Program Counter (PC) pointe sur la prochaine instruction à être exécutée
- Données statiques: Variables globales/constantes/statiques partagées entre tâches/threads
- Heap :Dynamique allouée avec malloc/free (C) ou new/delete (C++)
- Stack
  - Utilisé pour exécuter du code, méthode/fonction appel et retour
  - Position est dans le stack Pointer

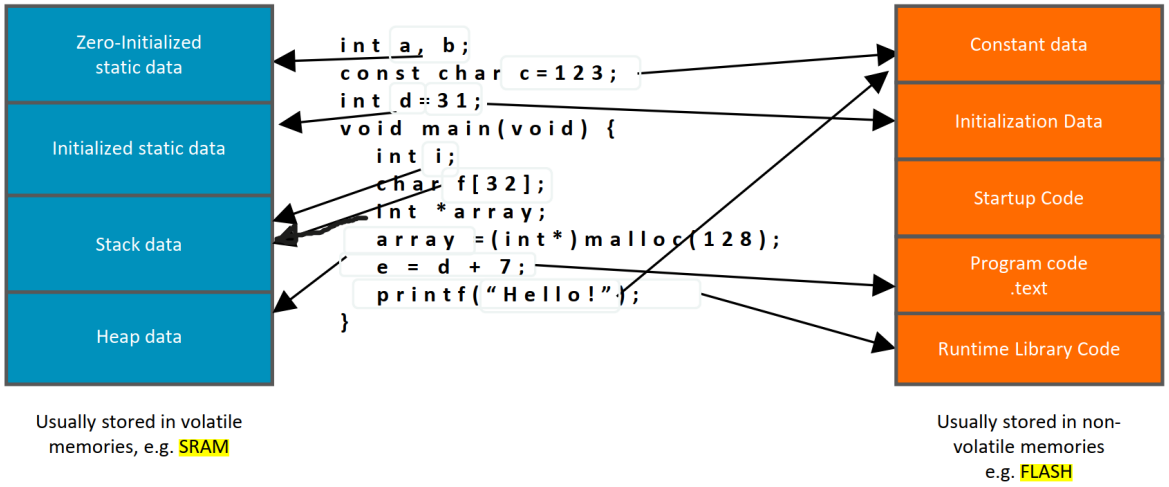
1.1 Compiler Stages

- Pre-processing
  - Remplace les macros qui sont définie par un # dans le code
  - Fusionne les sous-fichiers (.c/.cpp, .h) en un seul fichier (remplace les includes .h par son texte interne)
- Parser (analyseur)
  - Lit le code C
  - Vérifie les erreurs de syntaxes
  - Créer un code intermédiaire (représentation en arbre)
- Optimisateur Haut-niveau : modifie le code intermédiaire (indépendant du processeur)
- Générateur de code (dès qu'on connait l'architecture)
  - Crée le code assembleur étapes par étapes pour chaque noeud du code intermédiaire
  - Alloue les registres à différentes utilisations
- Optimisateur Bas-niveau : modifie le code assembleur (les parties sont spécifiques au processeur)
- Linker/Loader : créer une image exécutable du fichier objet

1.2 Model mémoire



1.3 Emplacement des données



1.4 Types de données

- const : fixe en ROM
- volatile : modifiable ISR pas d'optimisation
- static : garde leurs valeurs entre deux appelles

1.5 Hiérarchie mémoire (plus au moins rapide)

- registres : dans le CPU
- cache : (static RAM)
- main memory : dynamic RAM, volatile data
- secondary memory : flash/hard disk
- tertiary memory : tape libraries

1.6 Memory Protection Unit

Améliore la sécurité peut définir 8 régions avec différents accès

- unité programmable
- gère les accès mémoires et alloue des privilèges
- monteur les transactions (data, instructions)
- active un FAULT_EXCEPTION lors d'une violation d'accès

- (OS) définit régions mémoires
- (OS) définit les permissions pour y accéder