Introducción a FreeRTOS

Electrónica IV

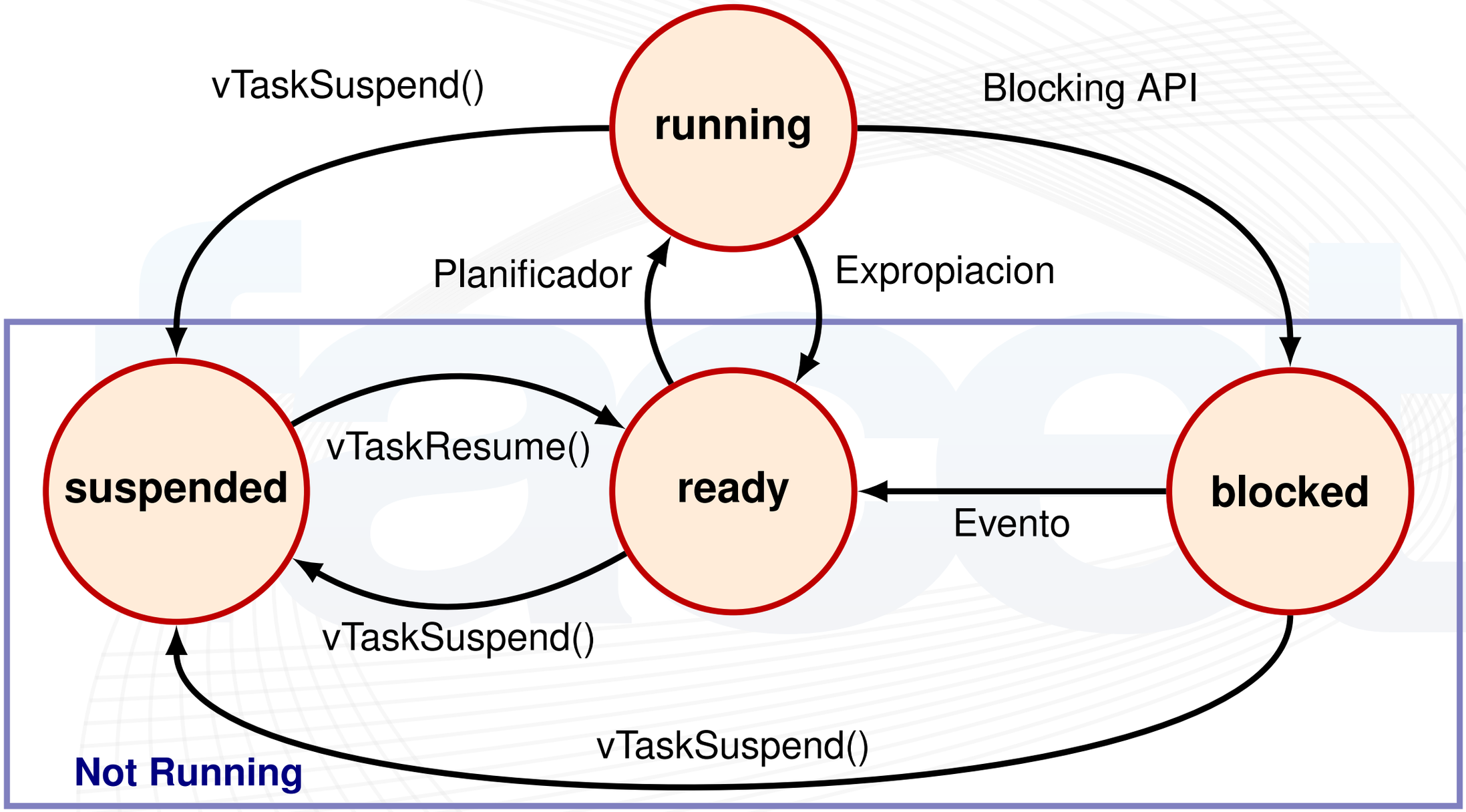
Mg.Ing. Esteban Volentini (evolentini@herrera.unt.edu.ar)

https://facetvirtual.facet.unt.edu.ar/course/view.php?id=165

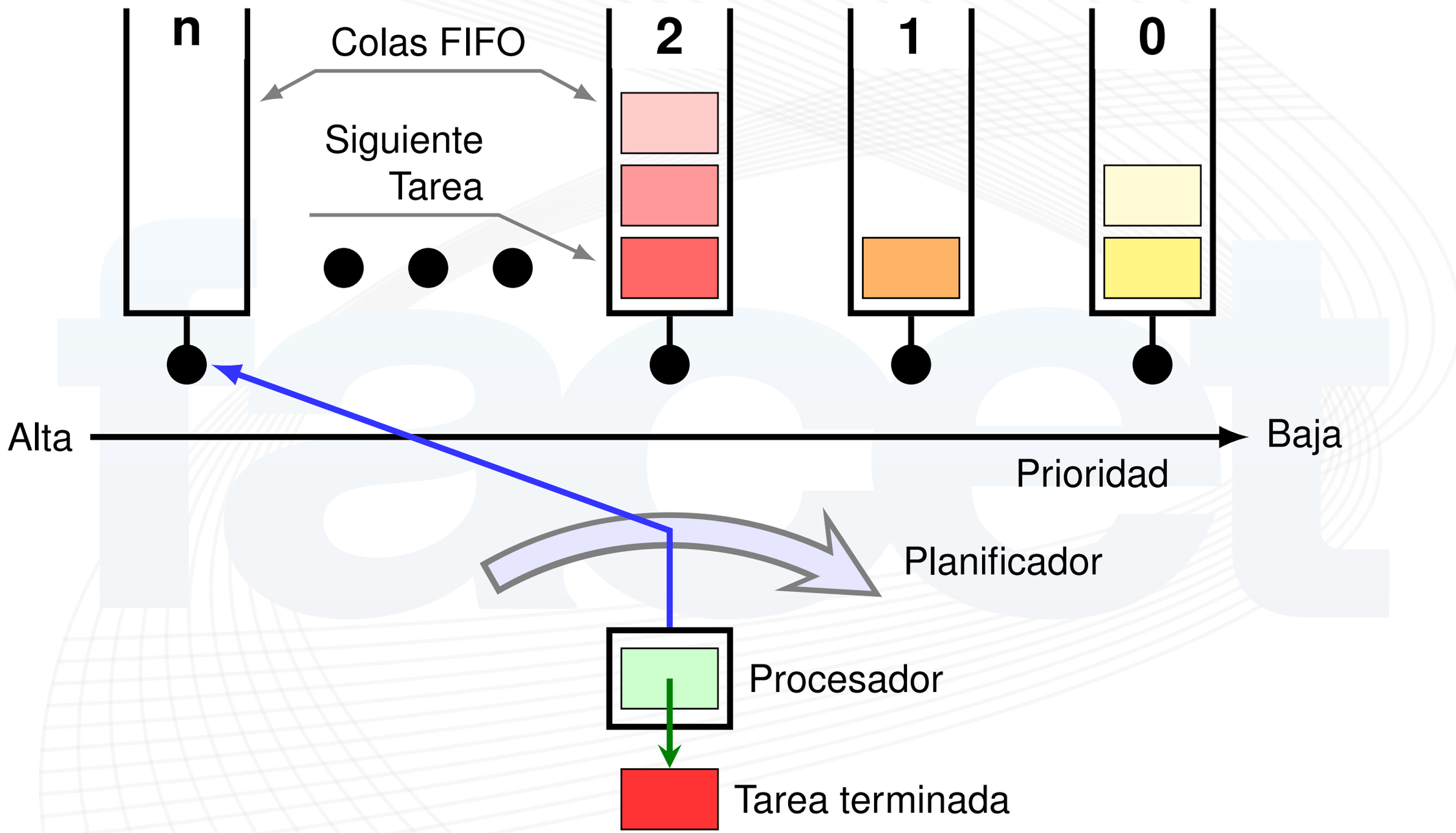
Sistema Operativo FreeRTOS

- Es un sistema operativo comercial con licencia para ser utilizado sin costo en aplicaciones comerciales
- Es un sistema operativo dinámico: las tareas son creadas en tiempo de ejecución, antes o después de iniciar el planificador
- El manejo de memoria puede ser estático o dinámico
- Implementa un planificador mixto que permite realizar un round-robin entre todas las tareas de igual prioridad

Estados en FreeRTOS



Política del Planificador FIFO



Una tarea en FreeRTOS

- Se definen como funciones que reciben un puntero como parámetro
- Tienen la estructura que habitualmente tiene la función `main()`, bloque secuencial seguido de lazo infinito.
- La ejecución no debe alcanzar la llave de cierre de la función, antes se debe pedir el sistema operativo la terminación de la tarea

Gestion de las de tareas

- `xTaskCreate[Static]()`
  - Crea una nueva tarea
- `vTaskDelete()`
  - Termina una tarea
- `vTaskSuspend()`
  - Suspende una tarea
- `xTaskResume[FromISR]()`
  - Reanuda una tarea suspendida
- `vTaskPrioritySet()`
  - Cambia la prioridad de una tarea

Tareas con parámetros

- Las funciones reciben un puntero como parámetro de entrada
  - Este puntero se asigna al crear la tarea
- El código debe ser reentrante:
  - Se debe poder empezar a ejecutar una segunda copia antes de terminar la primera
  - En general no se deben usar variables globales o estáticas

Tareas con parámetros

```c
/** @brief Estructura de parámetros para blinking */
typedef struct {
   uint8_t  led;   /** < Led que debe parpadear */
   uint16_t delay; /** < Demora entre cada encendido */
} blinking_t;

/** @brief Función que implementa la tarea blinking */
void Blinking(void * parametros) {
   blinking_t * valores = parametros;
   while (1) {
      Led_Toggle(valores->led);
      vTaskDelay(valores->delay / portTICK_PERIOD_MS);
   }
}
```

Tareas con parámetros

```
int main(void) {
    /* Variable con los parámetros de las tareas */
    static const blinking_t valores[] = {
        {.led = RED_LED, .delay = 500},
        {.led = RGB_B_LED, .delay = 300},
    };
    ...
    /* Creación de las tareas */
    xTaskCreate(Blinking, "Rojo", configMINIMAL_STACK_SIZE,
        (void *)&valores[0], tskIDLE_PRIORITY + 1, NULL);
    xTaskCreate(Blinking, "Azul", configMINIMAL_STACK_SIZE,
        (void *)&valores[1], tskIDLE_PRIORITY + 2, NULL);
    ...
}
```

Temporización en FreeRTOS

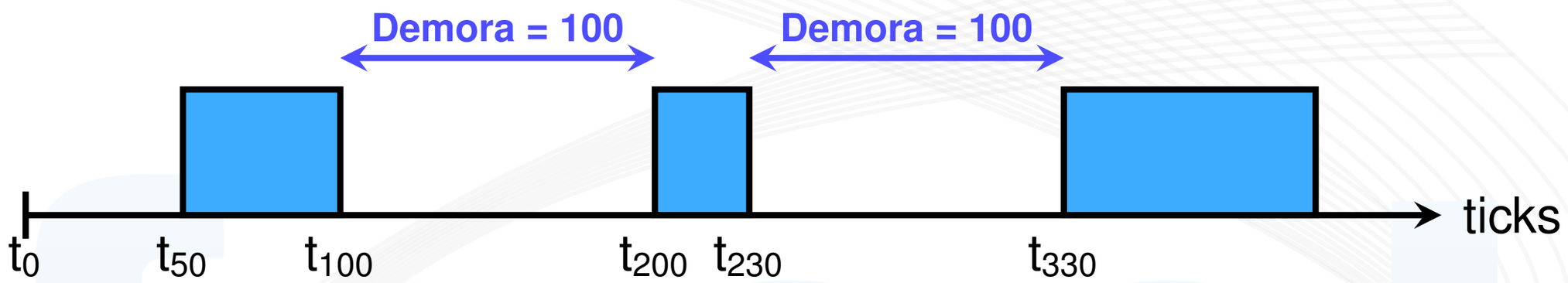
- Para la gestión del tiempo FreeRTOS ofrece esperas y temporizadores
- Las esperas son un mecanismo nativo del sistema operativo
- Bloquean la tarea por un cantidad de tiempo
- Al terminar la espera la tarea vuelve al estado ready

Manejo de esperas

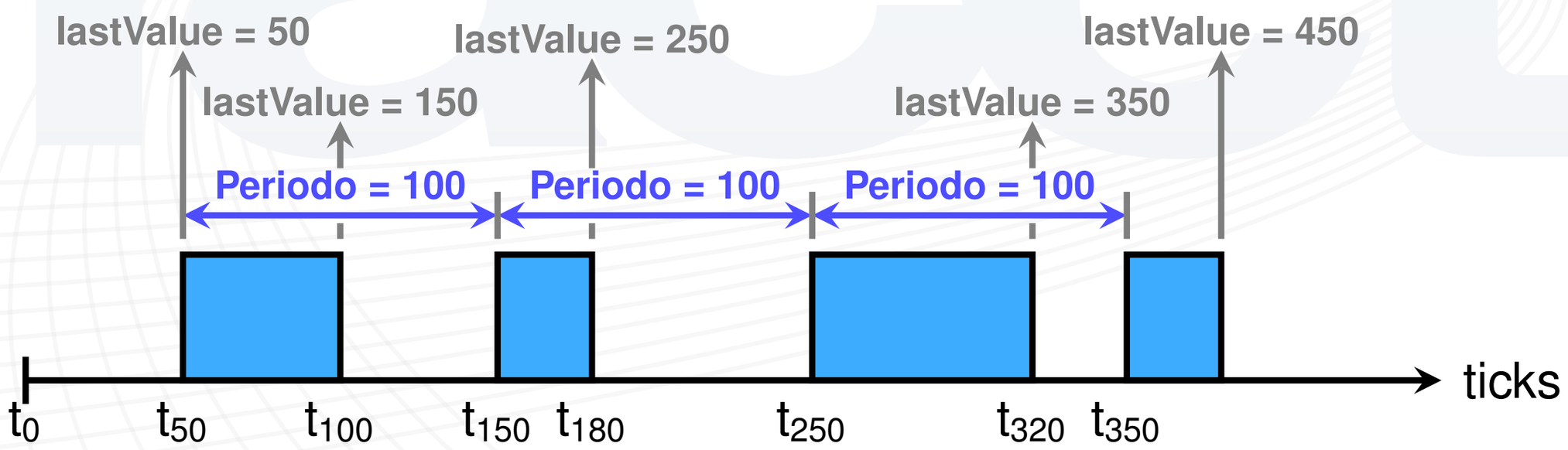
- **`vTaskDelay()`**
  - Suspende una tarea por un tiempo
- **`vTaskDelayUntil()`**
  - Suspende hasta alcanzar un valor
- **`xTaskAbortDelay()`**
  - Cancela la espera de una tarea

Diferencias entre las esperas

`xTaskDelay(pdMS_TO_TICKS(100))`



`xTaskDelayUntil(&lastValue, pdMS_TO_TICKS(100))`



Temporizadores de FreeRTOS

- Los temporizadores permiten programar una cuenta regresiva, periódica o de única vez
- Cuando la cuenta llega a cero se llama a una función de callback definida al crear el temporizador
- La implementación de los temporizadores utiliza una tarea interna llamada TimerTask
- Las funciones de callback se ejecutan secuencialmente en el contexto de TimerTask
- Para la comunicación con la tarea del usuario se utiliza una cola de mensajes que solo puede ser accedida usando las función de la API de temporizadores

Temporizadores FreeRTOS

- Se deben configurar las siguiente constantes en el archivo `FreeRTOSConfig.h`
- `configUSE_TIMERS = 1` para habilitar los temporizadores, lo que implica la creación automática de la tarea TimerTask
- `configTIMER_TASK_PRIORITY` para definir la prioridad de la tarea TimerTask
- `configTIMER_QUEUE_LENGTH` para definir la cantidad de entradas en la cola de comunicación
- `configTIMER_TASK_STACK_DEPTH` para definir la cantidad de bytes asignados a la pila de la tarea TimerTask

Temporizadores FreeRTOS

- **`xTimerCreate[Static]()`**
  - Crea un temporizador
- **`xTimerStart[FromISR]()`**
  - Inicia la cuenta regresiva de un temporizador
- **`xTimerReset[FromISR]()`**
  - Reinicia cuenta regresiva de un temporizador
- **`xTimerStop[FromISR]()`**
  - Detiene cuenta regresiva
- **`xTimerChangePeriod[FromISR]()`**
  - Cambia valor de la cuenta regresiva
- **`xTimerIsTimerActive()`**
  - Verifica si un temporizador está activo